

## PATENT ABSTRACTS OF JAPAN

(11)Publication number : 09-212359

(43)Date of publication of application : 15.08.1997

(51)Int. Cl.

G06F 9/38

(21)Application number : 08-249650

(71)Applicant : HITACHI LTD

(22)Date of filing : 20.09.1996

(72)Inventor : PURASENJITSUTO BISUWASU  
GAUTAMU DEWAN  
KEBIN IADONATO  
NAKAGAWA NORIO  
UCHIYAMA KUNIO

(30)Priority

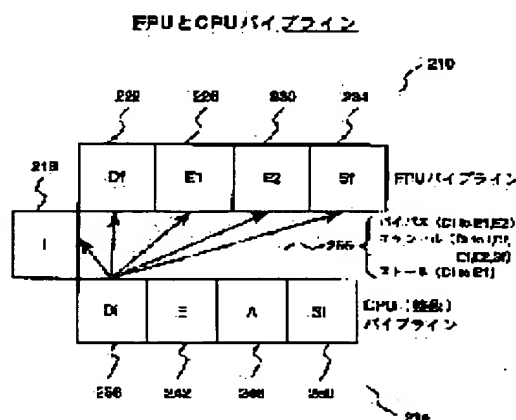
Priority number : 96 594763 Priority date : 31.01.1996 Priority country : US

## (54) MICROPROCESSOR AND DATA PROCESSING METHOD

(57)Abstract:

PROBLEM TO BE SOLVED: To reduce the physical circuit scale and to attain synchronization between an FPU pipeline and a CPU pipeline.

SOLUTION: The microprocessor executes an instruction by advancing the FPU pipeline 210 and the CPU pipeline 214 in parallel. When the number of stages in both the pipelines 210, 214 are basically set to the same number, resources for pipeline control such as the flow control of both the pipelines 210, 214 can be shared. When freezing or stalling is necessary for the pipelines 210, 214, one of the pipelines 210, 214 controls freezing or stalling so as to bring stalling and freezing effects to both the pipelines 210, 214, so that synchronization between the FPU pipeline 210 and the CPU pipeline 214 can be attained.



## LEGAL STATUS

[Date of request for examination]

26.02.2003

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision  
of rejection]

[Date of requesting appeal against examiner's  
decision of rejection]

[Date of extinction of right]

Copyright (C); 1998,2003 Japan Patent Office

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平9-212359

(43) 公開日 平成9年(1997) 8月15日

(51) Int.Cl.<sup>8</sup>

G 0 6 F 9/38

識別記号

3 1 0

庁内整理番号

F I

G 0 6 F 9/38

技術表示箇所

3 1 0 J

3 1 0 E

審査請求 未請求 請求項の数 8 O L (全 51 頁)

(21) 出願番号 特願平8-249650

(22) 出願日 平成8年(1996) 9月20日

(31) 優先権主張番号 5 9 4 7 6 3

(32) 優先日 1996年1月31日

(33) 優先権主張国 米国 (U S)

(71) 出願人 000005108

株式会社日立製作所

東京都千代田区神田駿河台四丁目6番地

(72) 発明者 ブラセンジット ビスワス

アメリカ合衆国 カリフォルニア州

95070、サラトガ、ハンパス コート  
20167

(72) 発明者 ガウタム デワン

アメリカ合衆国 カリフォルニア州

95014、クーバーティノ、ホームステッド  
ロード 20800 エービーティー #15  
ケー

(74) 代理人 弁理士 玉村 静世

最終頁に続く

(54) 【発明の名称】 マイクロプロセッサ及びデータ処理方法

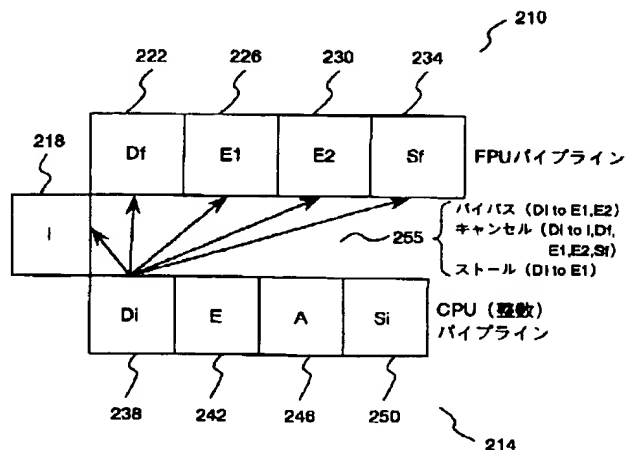
(57) 【要約】

【課題】 物理的な回路規模を縮小してFPUパイプラインとCPUパイプラインとを同期化する。

【解決手段】 マイクロプロセッサは、FPUパイプライン(210)とCPUパイプライン(214)を並列的に進めて命令を実行する。双方のパイプライン段数を基本的に同一とすることにより、双方のパイプラインのフロー制御などのパイプライン制御のためのリソースを共有化する。パイプラインに対してフリーズやストールを必要とする場合、双方のパイプラインにストールとフリーズの効果をもたらしようにパイプラインの一方がストール又はフリーズを制御することなどによってFPUパイプラインとCPUパイプラインの同期化を達成する。

【図2】

FPUとCPUパイプライン



# 【特許請求の範囲】

【請求項 1】 浮動小数点処理を実行する浮動小数点パイプラインと整数処理及びメモリアドレッシング処理を実行する整数パイプラインとを用いて命令を実行する浮動小数点ユニット及び中央処理装置を含んで成るマイクロプロセッサであって、

前記浮動小数点パイプラインのパイプライン段数と前記整数パイプラインのパイプライン段数とが等しくされて成るものであることを特徴とするマイクロプロセッサ。

【請求項 2】 前記浮動小数点パイプライン及び整数パイプラインは共通の命令フェッチステージを有し、前記浮動小数点パイプラインは更に、デコードステージ、第 1 の実行ステージ、第 2 に実行ステージ及びライトバックステージを有し、前記整数パイプラインは更に、デコードステージ、実行ステージ、メモリアクセスステージ及びライトバックステージを有して成るものであることを特徴とする請求項 1 記載のマイクロプロセッサ。

【請求項 3】 前記整数パイプラインのデコードステージは、浮動小数点パイプライン及び整数パイプラインのためのストール信号を生成するものであることを特徴とする請求項 1 記載のマイクロプロセッサ。

【請求項 4】 前記浮動小数点ユニットと中央処理装置は、浮動小数点パイプラインと整数パイプラインとのフロー処理を制御する論理を共有するものであることを特徴とする請求項 1 記載のマイクロプロセッサ。

【請求項 5】 浮動小数点ユニットと中央処理装置とを含むマイクロプロセッサにおいて、浮動小数点処理を実行する浮動小数点パイプラインと整数及びメモリアドレッシング処理を実行する整数パイプラインとを用いて命令を実行するデータ処理方法であって、

前記浮動小数点パイプラインのパイプライン段数は前記整数パイプラインのパイプライン段数に等しくされ、前記浮動小数点パイプラインと整数パイプラインは共通の命令フェッチステージを有し、

中央処理装置のレジスタから浮動小数点ユニットのレジスタにデータを転送する第 1 の命令を前記命令フェッチステージでフェッチする処理と、

整数パイプラインが前記第 1 の命令をデコードして中央処理装置のレジスタの値をバスに出力する処理と、

浮動小数点パイプラインが前記第 1 の命令をデコードして、前記出力する処理と同期的に前記バスの値を浮動小数点ユニットのレジスタに書き込む処理と、を含むことを特徴とするデータ処理方法。

【請求項 6】 浮動小数点ユニットと中央処理装置とを含むマイクロプロセッサにおいて、浮動小数点処理を実行する浮動小数点パイプラインと整数及びメモリアドレッシング処理を実行する整数パイプラインとを用いて命令を実行するデータ処理方法であって、

前記浮動小数点パイプラインのパイプライン段数は前記整数パイプラインのパイプライン段数に等しくされ、

前記浮動小数点パイプラインと整数パイプラインは共通の命令フェッチステージを有し、

浮動小数点ユニットのレジスタから中央処理装置のレジスタにデータを転送する第 2 の命令を前記命令フェッチステージでフェッチする処理と、

浮動小数点パイプラインが前記第 2 の命令をデコードして浮動小数点ユニットのレジスタの値をバスに出力する処理と、

整数パイプラインが前記第 2 の命令をデコードして、前記出力する処理と同期的に前記バスの値を中央処理装置のレジスタに書き込む処理と、を含むことを特徴とするデータ処理方法。

【請求項 7】 浮動小数点ユニットと中央処理装置とを含むマイクロプロセッサにおいて、浮動小数点処理を実行する浮動小数点パイプラインと整数及びメモリアドレッシング処理を実行する整数パイプラインとを用いて命令を実行するデータ処理方法であって、

前記浮動小数点パイプラインのパイプライン段数は前記整数パイプラインのパイプライン段数に等しくされ、

前記浮動小数点パイプラインと整数パイプラインは共通の命令フェッチステージを有し、前記浮動小数点パイプラインは更に、デコードステージ、第 1 の実行ステージ、第 2 に実行ステージ及びライトバックステージを有し、前記整数パイプラインは更に、デコードステージ、実行ステージ、メモリアクセスステージ及びライトバックステージを有し、

比較結果の一致／不一致に応じた論理値の T ビットを生成する浮動小数点比較命令に対して浮動小数点パイプラインが前記第 1 の実行ステージで T ビットを生成する処理と、

T ビットの値を分岐条件とする条件分岐命令に対して整数パイプラインがその実行ステージで前記 T ビットを参照する処理と、を含むことを特徴とするデータ処理方法。

【請求項 8】 浮動小数点ユニットと中央処理装置とを含むマイクロプロセッサにおいて、浮動小数点処理を実行する浮動小数点パイプラインと整数及びメモリアドレッシング処理を実行する整数パイプラインとを用いて命令を実行するデータ処理方法であって、

前記浮動小数点パイプラインのパイプライン段数は前記整数パイプラインのパイプライン段数に等しくされ、

前記浮動小数点パイプラインと整数パイプラインは共通の命令フェッチステージを有し、前記浮動小数点パイプラインは更に、デコードステージ、第 1 の実行ステージ、第 2 に実行ステージ及びライトバックステージを有し、前記整数パイプラインは更に、デコードステージ、実行ステージ、メモリアクセスステージ及びライトバックステージを有し、

前記第 1 の実行ステージを複数回繰り返す浮動小数点除算命令を命令フェッチステージでフェッチする処理と、

フェッチされた前記浮動小数点除算命令における第1の実行ステージの期間にビジー信号をイネーブルにする処理と、

前記ビジー信号がイネーブルにされている期間において、浮動小数点パイプラインを利用する後続の命令の第1の実行ステージ以降のパイプラインステージと整数パイプラインのステージとを共に同期させてストールする処理と、を含むことを特徴とするデータ処理方法。

#### 【発明の詳細な説明】

##### 【0001】

【発明の属する技術分野】本発明は、浮動小数点ユニット及び中央処理装置を含んだマイクロプロセッサ、特にそれにおけるパイプライン制御技術に関し、例えば、RISC形式のマイクロプロセッサに適用して有効な技術に関するものである。

##### 【0002】

【従来の技術】あるRISC（縮小命令セットコンピュータ）マイクロプロセッサは、FPU（浮動小数点ユニット）を備える。浮動小数点ユニットは、浮動小数点計算を実行する回路である。RISCは、相対的に単純で、固定サイズの命令を用いて、マイクロプロセッサの複雑さを減少させたコンピュータアーキテクチャである。RISCアーキテクチャにおける殆どの命令は、汎用レジスタを利用してオペランドの操作を行ない、結果をレジスタにストアする。これらのレジスタは、メモリからロードされ、典型的には、レジスタの内容はプログラムの実行中に再利用される。殆どのRISCアーキテクチャは16又はそれ以上の汎用レジスタを持つ。

##### 【0003】

【発明が解決しようとする課題】典型的なRISCマイクロプロセッサは、パイプラインで命令を実行する能力を持つ。多数の機能ユニット（CPUの整数処理などを実行するCPUパイプラインや浮動小数点処理を実行するFPUパイプライン）の動作を調整することにおいて多数の問題がある。そのような装置における二つのユニット（CPUパイプラインを制御するユニットとFPUパイプラインを制御するユニット）がパイプライン制御のためのリソースを共有するなら、二つのパイプラインの動作を同期化させることは、その問題の解決に大きな役割を演ずるであろうことが本発明者によって明らかにされた。すなわち、FPUとCPUを含むマイクロプロセッサについて考える。従来、FPUパイプライン及びCPUパイプラインの夫々の段数（パイプラインステージの数）は相違されている。浮動小数点演算は整数演算に比べて演算時間がかかるので、実行ステージに3段のパイプラインステージを割り当てたものがある。FPUパイプラインとCPUパイプラインの段数が相違されると、ストールやバイパスなどのフロー制御を行うためのデコードロジック等はCPU及びFPU双方のパイプライン毎に別々に構成することが必要になる。例えば命令

毎のパイプライン間でのバイパスを一例として考えると、前後の命令でレジスタコンフリクトのようなデータコンフリクトを生ずるとき、前の命令が演算を完了したとき、その実行ステージの結果をバイパス経路にて後の命令の実行ステージに直接与えることによって当該後の命令は先の命令の実行完了を待つことなくオペランドを得ることができるが、どのステージからどのステージにバイパスさせるべきかは、パイプラインステージの段数に依存して決定されなければならない。このとき、FPUパイプラインとCPUパイプラインの長さ（パイプライン段数）が相違すれば、バイパス制御などのフロー制御のためのリソースもCPUパイプラインとFPUパイプラインとの間で殆ど個別化することが必要になり、また、FPUパイプラインとCPUパイプラインとの間でのデータ交換に際しても複雑なハードウェアが必要になってしまう。本発明者はそれら点について見出した。

【0004】もう一つの問題は、正確に例外の意味を維持することである。パイプライン化されたアーキテクチャ又はマルチ・ファンクションユニット・アーキテクチャにおいて例外又は割り込みを正確に（プログラムの実行順序を乱さずに）処理することは、例外又は割り込みが発生したとき、アーキテクチャ上連続的に実行されるプログラムと正確に同じになるように、マシンの状態をセーブ可能であるべきことを意味する。たとえ中継ユニットに指示を発行する命令が厳密にプログラムの順序を維持するとしても、命令の実行順序は、異なった機能ユニットでは命令の実行時間が異なるため、乱れる。例えば、CPUパイプライン及びFPUパイプラインを有するプロセッサにおいて、浮動小数点除算命令のような通常よりも実行サイクル数の多い命令をFPUパイプラインで処理するとき、その処理で0除算のような例外が発生したとき、当該浮動小数点除算命令の後に続く命令の実行状態如何によっては、例外発生に起因して当該浮動小数点除算命令を再実行すると、命令実行順序がプログラムで規定されている命令実行順序を乱して、制御が複雑になる場合がある。

【0005】パイプライン化されたプロセッサにおいて上記の意味で正確に割り込みを実施するための効果的な手段は、先行技術としてのIEEEコンピュータ処理、第562～573頁の「パイプライン化されたプロセッサにおける厳密な割り込み」（“Implementing Precise Interrupts in Pipelined Processor” IEEE Transactions on Computers, pp. 562-573, May 1988）で議論されている。現在のパイプライン化された多くの機能ユニットを有するプロセッサの殆どは、そこで言及されている技術に基づいている。

【0006】それら技術の内のあるものは、レジスタファイルの追加を必要とし、制御のための複雑なロジックを必要とする。リソースを共有して同期化するには、共有リソースに結合された更に複雑な内部データベースと同

様に、タグ・マッチングのハードウェアを必要とする。他の技術は、レジスタリソースのコンフリクト（競合）を識別し且つ解消するためのレジスタスコアボードを使用する。これらの技術は、要するに、チップエリアの追加を要求し、エンベデッド・アプリケーション（機器組み込み制御）を予定する低価格のプロセッサには適していない。

【0007】本発明の目的は、浮動小数点パイプラインと整数パイプラインとを同期化させることができるマイクロプロセッサを提供することにある。

【0008】更に本発明は、上記同期化に際してパイプライン制御のためのリソースを低減出来るマイクロプロセッサを提供することにある。

【0009】本発明の別の目的は、CPUとFPUとの間のデータ交換を簡単にしかも高速に行なうことができるデータ処理方法を提供することにある。

【0010】本発明の更に別の目的は、浮動小数点比較命令の後に続く条件分岐に対するスループットを向上させることができるデータ処理方法を提供することにある。

【0011】本発明のその他の目的は、浮動小数点除算命令においても浮動小数点パイプラインと整数パイプラインとを同期化できるデータ処理方法を提供することにある。

【0012】本発明の前記並びにその他の目的と新規な特徴は本明細書の記述及び添付図面から明らかになるであろう。

【0013】

【課題を解決するための手段】本願において開示される発明のうち代表的なものの概要を簡単に説明すれば下記の通りである。

【0014】すなわち、マイクロプロセッサ（110）は、浮動小数点処理を実行する浮動小数点パイプライン（FPUパイプライン）、そして、整数計算及びメモリアドレス操作を実行する整数パイプライン（CPUパイプライン）を利用して命令を実行する。すなわち、命令実行に際して、FPUパイプラインとCPUパイプラインを並列的に進めていく。浮動小数点パイプライン（210）とCPUパイプライン（214）とのパイプライン段数（パイプラインの長さ）を基本的に同一とすることによって、CPUパイプラインとFPUパイプラインのフロー制御などのパイプライン制御のためのリソースの共有化を図る。これによって、浮動小数点パイプラインと整数パイプラインとを同期化し易くする。パイプラインに対してフリーズやストールを必要とする場合、双方のパイプラインにストールとフリーズの効果をもたらすようにパイプラインの一方がストール又はフリーズを制御することによってFPUパイプラインとCPUパイプラインの同期化を達成する。

【0015】例外は、長い浮動小数点処理でも正確に保

たれる。正確な例外は、プログラムが規定する命令実行順序における先の浮動小数点命令である浮動小数点除算命令がFPUパイプラインの最初の実行ステージに入るとき、FPUパイプラインステージの最初の実行ステージでビジー信号をアサートする（イネーブルにする）ことによって達成される。前記先の浮動小数点命令がFPUパイプラインの最初の実行ステージを繰返し利用する複数ステージ分の時間を費やした実行を完了する前に、後の浮動小数点命令がFPUパイプラインによってデコードされる場合、それ以降のCPU及びFPUの双方のパイプラインがストールされる。

【0016】本発明による手段を更に説明する。

【0017】浮動小数点処理を実行する浮動小数点パイプライン（210）と整数処理及びメモリアドレス処理を実行する整数パイプライン（214）とを用いて命令を実行する浮動小数点及び中央処理装置を含むマイクロプロセッサは、前記浮動小数点パイプラインのパイプライン段数と前記整数パイプラインのパイプライン段数とが等しくされて成る。双方のパイプライン（210、214）の段数が等しければ、ストール及びフリーズを制御するフロー制御等のためのリソースを双方のパイプラインで共有させることができる。これは、チップ面積の縮小を可能にする。フロー制御のためのリソースの共有は双方のパイプラインの同期化を容易にする。

【0018】前記整数パイプラインのデコードステージは、浮動小数点パイプライン及び整数パイプラインのためのストール信号を生成することができる。

【0019】浮動小数点ユニット及び中央処理装置を含むマイクロプロセッサにおいて、浮動小数点処理を実行する浮動小数点パイプラインと整数処理及びメモリアドレス処理を実行する整数パイプラインとを用いて命令を実行するデータ処理方法の観点では、前記浮動小数点パイプラインのパイプライン段数は前記整数パイプラインのパイプライン段数に等しくされ、前記浮動小数点パイプラインと整数パイプラインは共通の命令フェッチステージを有する。このとき、ある命令を実行するときにおけるFPUパイプラインとCPUパイプラインとの同期化の例としては、中央処理装置のレジスタから浮動小数点ユニットのレジスタ（FPU L）にデータを転送する第1の命令によるデータ処理方法がある。この第1の命令を実行するデータ処理方法は、この命令を前記命令フェッチステージでフェッチする処理と、整数パイプラインが前記第1の命令をデコードして中央処理装置のレジスタの値をバスに出力する処理と、浮動小数点パイプラインが前記第1の命令をデコードして前記出力する処理と同期的に前記バスの値を浮動小数点ユニットのレジスタに書き込む処理とを含む。

【0020】また、上記同期化の別の例として、浮動小数点ユニットのレジスタから中央処理装置のレジスタにデータを転送する第2の命令によるデータ処理方法を挙

げることができる。この第2の命令を実行するデータ処理方法は、当該第2の命令を前記命令フェッチステージでフェッチする処理と、浮動小数点パイプラインが前記第2の命令をデコードして浮動小数点ユニットのレジスタ(FPUL)の値をバスに出力する処理と、整数パイプラインが前記第2の命令をデコードして前記出力する処理と同期的に前記バスの値を中央処理装置のレジスタに書き込む処理とを含む。

【0021】前記浮動小数点パイプラインのパイプライン段数を前記整数パイプラインのパイプライン段数に等しくし、前記浮動小数点パイプラインと整数パイプラインが共通の命令フェッチステージを有する場合の前記データ処理方法において、前記浮動小数点パイプラインは更に、デコードステージ、第1の実行ステージ、第2に実行ステージ及びライトバックステージを有し、前記整数パイプラインは更に、デコードステージ、実行ステージ、メモリアクセスステージ及びライトバックステージを有するとき、浮動小数点比較命令に後続する条件分岐命令による条件分岐のスループットを向上させることを企図したデータ処理方法は、比較結果の一致/不一致に応じた論理値のTビットを生成する浮動小数点比較命令に対して浮動小数点パイプラインが第1の実行ステージでTビットを生成する処理と、Tビットの値を分岐条件とする条件分岐命令に対して整数パイプラインがその実行ステージで前記Tビットを参照する処理とを含む。これによって、浮動小数点比較命令の直後に条件分岐命令が配置されている場合にストールは一切不要である。

【0022】前記浮動小数点パイプラインのパイプライン段数を前記整数パイプラインのパイプライン段数に等しくし、前記浮動小数点パイプラインと整数パイプラインが共通の命令フェッチステージを有する場合のデータ処理方法において、前述と同じく、前記浮動小数点パイプラインは更にデコードステージ、第1の実行ステージ、第2に実行ステージ及びライトバックステージを有し、前記整数パイプラインは更にデコードステージ、実行ステージ、メモリアクセスステージ及びライトバックステージを有するとき、浮動小数点除算命令によるFPUパイプラインとCPUパイプラインとの同期化を保つようにするデータ処理方法は、第1の実行ステージを複数回繰り返す浮動小数点除算命令を命令フェッチステージでフェッチする処理と、フェッチされた前記浮動小数点除算命令における第1の実行ステージの期間にビジー信号をイネーブルにする処理と、前記ビジー信号がイネーブルにされている期間において、浮動小数点パイプラインを利用する後続の命令の第1の実行ステージ以降のパイプラインステージと整数パイプラインのステージとを共に同期させてストールする処理とを含む。

【0023】

【発明の実施の形態】本発明の好適な例において、マイクロプロセッサは、以下に説明されるように、浮動小数

点処理を実行するために浮動小数点パイプライン(FPUパイプライン)210(図2参照)を利用し、整数及びメモリアドレッシング処理などを実行するために整数パイプライン(CPUパイプライン)214(図2参照)を利用する。FPUパイプライン210とCPUパイプライン214とは並列的に進行され、その過程で同期化が行われる。FPUパイプライン210とCPUパイプライン214のパイプライン段数(パイプラインのステージ数)を相互に等しくしてある。これによって、CPUパイプライン214とFPUパイプライン210とのフロー制御等のためのパイプライン制御用のリソースがFPUパイプラインとCPUパイプラインで殆ど共有出来るようになる。このパイプラインリソースの共有は、CPUパイプライン214とFPUパイプライン210との同期化を容易にする。

【0024】FPUパイプライン210は幾つかの点でCPUパイプライン214に同期化される。FPU及びCPUパイプライン210、214は互いに同期化され、例えば、浮動小数点のロード/ストア/リストア命令のような命令が、FPU114とCPU118がリソース(例えばコミュニケーションレジスタ)を共有するときである。FPUとCPUパイプライン210、214は、それらがコミュニケーションレジスタFPULを使ってデータ交換を行なうとき同期化される。また、命令の手順が、例えばCPU分岐命令が後に続く浮動小数点比較命令においても、パイプラインの同期化が行われる。これら二つのパイプライン210、214の同期化は、更に以下で説明するように、正確な例外を保つにも有効である。

【0025】図1は本発明の好適な一例に係るプロセッサのブロックダイアグラムを示す。プロセッサ110は浮動小数点ユニット(FPU)114を持つ。更に、プロセッサ110は、整数を操作することができる中央処理装置(CPU)118を備える。プロセッサ110は、16ビット固定長浮動小数点命令セットを備えた32ビットRISCアーキテクチャとされる。CPU118のための16ビット命令の詳細は、論文、すなわち“SH3、高コード密度及びローパワー”IEEE Micro, 第11~19、12月、1995(“SH3:High Code Density, Low Power”, IEEE Micro, pp.11-19, December 1995)に述べられている。前記CPU118は、32ビットデータバス122を経由してFPU114に結合されている。整数の累積乗算(積和演算)ユニット(IMAC)120がデータバス122に結合されている。図1の回路間のインタフェース信号は、CPU118とFPU114との間のインタフェース信号124、125以外は図示を省略してある。前記CPU118は、32ビット論理アドレスバス126を経由してメモリマネージメントユニット装置(TLBコントローラ)134に結合されている。TLBは先見型アドレス

変換バッファを表している。TLBコントローラ134は命令とデータの混在型のTLB138を制御する。そのTLB138は32ビット物理アドレスバス130を介してキャッシュコントローラ142に接続されている。前記キャッシュコントローラ142は命令とデータを混在させたキャッシュメモリ146を制御する。キャッシュメモリ146は32ビットデータバス122を介してCPU118とFPU114に結合される。

【0026】マイクロプロセッサ110は更に、周辺データバス150及び周辺アドレスバス151に接続されたシリアルコミュニケーションインタフェース152、タイマ153等を備え、それら周辺データバス150及び周辺アドレスバス151はバスステートコントローラ154を介して前記データバス122及び物理アドレスバス130にインタフェースされる。内外からの割り込み要求の調停等は割り込みコントローラ155が行う。チップ外部とのバスインタフェースは外部バスインタフェース回路156によって行われる。この説明と図1に示された名称に基づけば、図1に示されているその他の回路の機能は当業者にとっては自ずと理解されるであろう。

【0027】プロセッサ110の機能は以下の説明によって理解されるであろう。FPU114は浮動小数点処理のためにデータや命令を必要とする。この例において、FPU114は、キャッシュメモリにデータを格納し或いはデータを獲得するためのメモリアドレス能力を持ち合わせていない。これは、FPUのメモリアドレス回路の必要性を取り除いてチップ面積を節約する。その代わりに、CPU118がFPU114に代わってキャッシュメモリをアドレスリングする。CPU118はFPU114のためにメモリからデータのフェッチを行なうだけでなく、FPU114のための浮動小数点命令を含む全ての命令をメモリからフェッチする。更にまた、以下により詳細を説明するように、FPU114とCPU118との間でのデータ転送は、メモリアクセスを伴わずに、代わりにFPUレジスタのようなレジスタを介して高速に行われる。

【0028】データ又は命令を獲得するために、CPU118は、論理アドレス（仮想アドレス）を計算し、32ビット論理アドレスバス126を介してメモリマネージメント装置134にその論理アドレスを送ってメモリ116からのデータ又は命令を要求する。

【0029】もしも、対応する物理アドレスが既にTLB138に格納されていないなら、TLBミスが発生し、TLBコントローラ134は、追加のマッピング情報を利用して論理アドレスを物理アドレスに変換するためのプログラムシーケンスを始める。メモリマネージメント装置134は、そのときTLB138に物理アドレスを格納する。CPU118が再び同じアドレス範囲のデータを要求するかもしれないので、TLB134は将

来の利用のためにそのアドレスを新たなエントリとして格納する。TLB138は32ビット物理アドレスバス130を介してキャッシュコントローラ142に物理アドレスを送る。キャッシュコントローラ142は32ビットデータバス122にデータ又は命令を置くように、データと命令が混在されるキャッシュメモリ146に指示する。仮に、要求されたアドレスがキャッシュメモリ146上で利用可能でないなら、キャッシュミスが発生し、CPU118とFPU114の処理は、要求された情報が外部メモリからキャッシュメモリにフェッチされるまで、キャッシュミス信号を適用することによってフリーズされる。ここで、フリーズとは、命令それ事態の配列に起因して生ずるようなメモリコンフリクトやレジスタコンフリクトのような状態に応じてパイプラインを止めるストールとは異なる概念を一般に持つ。すなわち、フリーズとは、命令の配列（命令の実行順序）に起因しない要因、例えば、キャッシュミスやTLBミスなどによってパイプラインをある期間止めることを意味する。ストールもフリーズも、ハザード（障害）に対処する一つの手段であることには変わらない。

【0030】命令はFPU114と同様CPU118によりデコーディングのために拾い上げられる。データはCPU118とFPU114によって共有される共通32ビットデータバス上で利用可能である。FPU114はデータ及び命令フェッチのためのメモリアドレスリングを行なう能力を持っていない。

【0031】CPU118とFPU114との間でのデータ交換は、専用のコミュニケーションレジスタFPUを介して行われる。上述のように、その他のRISCプロセッサにおいて、CPU118とFPU114との間のデータ交換はキャッシュメモリ146のような転送メモリ（transfer through memory）を介して起こる。FPUレジスタのようなレジスタを介する転送はキャッシュメモリ146を介する転送に比べて速い。

【0032】浮動小数点命令及び浮動小数点命令に関連するCPU命令のリストは図20に示されている。これら命令は図21～図52に詳細が記述され、命令のC言語記述を含んでいる。図21～図52に列挙されたそれら命令には、命令の意味に関係する重要性を持っていないアンダーラインが示されている。

【0033】図2にはFPU114とCPU118が命令を実行するのに利用するパイプラインを例示する。FPUパイプライン210とCPUパイプライン214は、単一の命令フェッチステージ218を共有する。更にFPUパイプライン210は、デコードステージ（Df）222、第1実行ステージ（E1）226、第2実行ステージ（E2）230、及びライトバックステージ（Sf）210の4ステージを更に持つ。CPUパイプライン214は、デコードステージ（Di）238、実行ステージ（E）242、メモリアクセスステージ（A24



6) 及びライトバックステージ(Si)250の、更に4ステージを持つ。Diステージ238は、図2の矢印255によって示されるようなFPUパイプラインステージに対してバイパス、ストール及びキャンセルのための信号を生成する。Diステージ238は、信号バス255を介して、E1、E2ステージ226、230にバイパス信号を供給し、全てのFPUステージ222、226、230、234と共有Iステージ218にキャンセル信号を供給し、ストールに関連する信号をDfステージに供給し、そしてストール信号をE1ステージに供給する。これらの信号は以下で更に説明する。

【0034】データバス122の上で利用出来る命令は、インストラクションフェッチステージ218によって最初獲得される。双方のデコードステージDf222及びDi238は、インストラクションフェッチステージでフェッチされた命令をデコードする。デコードステージの第1フェーズは、CPU命令かFPU命令かを識別することを含む。FPU命令は命令の上位4ビットにおけるF(16進数)によって識別される。命令が浮動小数点タイプでないなら、Dfステージ222はこれ以上命令をデコードしない。同様に、Diステージ238は浮動小数点命令を完全にデコードしない。Diステージ238は、実行されるべき浮動小数点機能を識別するために浮動小数点命令をデコードすることはしない。これはハードウェアの複雑性に対して縮小をもたらし、重要なことである。もしも単一のデコードステージだけが利用されるなら、FPUのデータバスを制御するために要求される信号は、CPU118からFPU114を交差しなければならず、チップ面積を増加させることになる。フェッチされた命令が浮動小数点命令であるとき、FPUパイプライン210のE1ステージ226は、命令の実行を始める。FPUパイプライン210のE2ステージ230は、その浮動小数点命令の実行を完了する。命令の必要条件に依存して、FPUパイプライン210のSfステージは、浮動小数点レジスタに命令の実行結果をストアすることができる。

【0035】フェッチされた命令が、整数を操作するための命令のような、CPU命令である場合、CPUパイプライン214のEステージ242は、当該命令を実行する。CPUパイプライン214のAステージ246は、実行されている特定の命令によってそれが要求されたとき、キャッシュメモリ146をアクセスする。最後に、CPUパイプライン214のSiステージ250は、命令実行結果を、例えばCPUレジスタ410の一つに書き込むことができる。有利なことに、FPUパイプライン210又はCPUパイプライン214の一つだけを利用することを要求する命令は、利用されないパイプラインに関しては単にそれを通過する命令、ということになる。例えば、命令フェッチステージ218が整数加算命令をフェッチしたとき、CPUパイプライン21

4は実行ステージ242で整数加算を実行し、Siステージ250でレジスタにその実行結果を格納する。しかし、整数加算命令がデコードされると、FPUパイプライン210のDfステージ222は整数加算命令をE1ステージ226に通過させる。整数加算命令は後の複数クロックサイクルの間、FPUパイプライン210の残りのステージを通過し続ける。同様に、フェッチされた命令が純粋な浮動小数点命令であるとき、前記Diステージ238は浮動小数点命令をEステージ242に通過させ、そして、後の複数サイクルの間、CPUパイプライン214の残りのステージを単に通過させる。ここで上記通過とは、表裏の関係にあるCPUパイプラインとFPUパイプラインにおいて実質的なオペレーションを伴わないパイプラインステージがそのパイプラインを単に通過していくということを意味する。

【0036】ここで、CPUパイプラインとFPUパイプラインとを図2に例示されるように構成することの意義を説明する。すなわち、FPUパイプライン210とCPUパイプライン214のパイプライン段数(パイプラインのステージ数)を相互に等しくしてある。これによって、CPUパイプライン214とFPUパイプライン210とのフロー制御等のためのパイプライン制御用のリソースがFPUパイプラインとCPUパイプラインで殆ど共有出来るようになる。このパイプラインリソースの共有によってパイプライン制御のための物理的な回路規模を縮小することができる。例えば、図18に示されるようなレジスタコンフリクトを生ずる場合を一例とすると、先の命令“Add R0, R12”のディスティネーションレジスタ(R1)が後の命令“Add R1, R2”のソースレジスタ(R1)である場合、先の命令のディスティネーションレジスタ(R1)に演算結果が得られた後でなければ、後の命令にとってそのレジスタ(R1)の値を参照することは無意味である。そのために、図示はしないが、パイプラインP1のライトバックSiステージでレジスタR1に演算結果が得られるまで、パイプラインP2をストールすることができる。このとき、パイプラインP1の実行ステージEで演算された演算結果をパイプラインP2の実行ステージEにバイパスして渡すことにより、ストールを行わずに済む。同様に、パイプラインP3～P5で示されるように、レジスタR11に関してデータコンフリクトを生ずる命令“Add R10, R11”と“Add R11, R12”との間に別の命令が介在されている場合には、パイプラインP3の実行ステージEで演算された演算結果をメモリアクセスステージAでパイプラインP5の実行ステージEにバイパスして渡すことにより、ストールを行わずに済む。上述のストールを行う期間、そして、上記バイパスをどのステージからどのステージに対して行うかという制御は、パイプラインステージの長さ(段数)に依存することが理解されるであろう。通常、ストール

の制御はストールされるべきパイプライン側で制御し、また、バイパス制御はバイパスされるデータの受け側で制御する。そうすると、FPUパイプラインとCPUパイプラインの段数が相違する場合には、バイパスに代表されるようなパイプライン間でのデータ交換、ストール、フリーズ、及びキャンセル等のための制御を行うリソース、即ち、パイプラインのフロー制御などを行うためのロジック回路やバイパスのための制御線などを、CPUパイプラインとFPUパイプラインとの間で殆ど個別化しなければならなくなる。この実施の形態に示される例では、図2に例示されるように、FPUパイプライン210とCPUパイプライン214のパイプライン段数を相互に等しくし、CPUパイプライン214とFPUパイプライン210とのフロー制御等のためのパイプライン制御用のリソースをFPUパイプラインとCPUパイプラインで殆ど共有させることができる。図19はCPUパイプラインとFPUパイプラインを制御するリソース、特にフロー処理のための論理の共有状態を概念的に示すものであり、CPU命令のためのフロー処理の論理がFPU命令のためのフロー処理に共有され、そのフロー処理の論理にはFPU命令のための付加ロジックは僅かに追加されるだけである。この付加ロジックはフロー制御の対象がFPUパイプラインかCPUパイプラインかを示すフラグなどとされる。

【0037】以下、CPUパイプラインとFPUパイプラインの種々の同期化の具体例を説明する。

【0038】ある命令はFPUパイプライン210とCPUパイプライン214との間でのデータ転送を要求する。そのような命令の一例として、CPUによるコミュニケーションレジスタ(FPUL)へのロード命令(LDSRm, FPUL)、すなわち、CPUロード・コミュニケーションレジスタ命令がある。この命令は、FPUパイプライン210とCPUパイプライン214並びにそれらパイプライン210、214の間でのデータ転送タイミングを示す図3で示されている。パイプラインの構成は図2で説明した通りであり。パイプラインの夫々のステージはまた、例えば一つのクロック308の位相(明らかなように二つのクロックの位相は示されていない)の単一のクロックサイクルに同期することに注意されたい。すなわち、一つのパイプラインステージはクロック308の1サイクルで実行される。FPULはFPU114に含まれるレジスタである。このレジスタFPULはデータ処理上、FPU110とCPU118が共有する。CPUによるコミュニケーションレジスタ(FPUL)へのロード命令はCPU命令である。しかしながら、上述のように、全ての命令はFPU及びCPUパイプライン210、214における双方のデコードステージ222、238でデコードされる。だから、CPUによるコミュニケーションレジスタ(FPUL)へのロード命令をデコードするや否や、Dfステージ22

2は、FPU114がレジスタFPULへのアクセスを制御することになるので、FPUパイプライン210を実質的に動作させるべきであると認識する。最初、CPUがコミュニケーションレジスタ(FPUL)にロードする命令は、CPUパイプライン214のEステージによって実行される。同時に、FPUパイプライン210のE1ステージ226は、文字(T)310によって示されるように、何ら動作することなく命令を通過させる。言い換えれば、CPUがコミュニケーションレジスタ(FPUL)にロードする命令は、E1ステージに対して単に通過することになる。

【0039】ここで、E1ステージに関連した別の説明を行う。パイプライン210、214の夫々のステージは、実行に1サイクルを要する。しかしながら、命令がパイプラインステージに1サイクルよりも多くの時間を費やすという、特別な場合がある。その場合に、命令はその特定のパイプラインステージを繰り返すことになる。例えば、図28及び図29に示されるような浮動小数点除算命令(FDIV)は、13サイクルのレーテンシ(待ち時間)を持つ。ここで、レーテンシは、命令が例えばFPUパイプライン210の実行ステージ226、230で費やすサイクル数の合計の尺度である。浮動小数点命令はE2ステージ230で1サイクルを費やす。それによって明らかなように、浮動小数点除算命令はE1ステージで12サイクルを費やす。図28には浮動小数点除算命令のピッチ(Pitch)が示され、それは一般に、現在の命令の後に続く命令がパイプラインで実行開始できる前のクロックサイクルの尺度である。例えば、浮動小数点除算命令のピッチは12サイクルに等しいから、浮動小数点除算命令の後に続く次の命令は12サイクルの後に実行することができる。ピッチの値12は、浮動小数点除算命令はE1ステージ226で12サイクル費やすということを示す。したがって、次の浮動小数点命令はE1ステージ226に入る前に、12クロックサイクル待たなければならない。

【0040】図3に示されるコミュニケーションレジスタ(FPUL)へのロード命令の例に戻ると、同じ命令がFPU及びCPUパイプライン210、214の双方に流れるので、E1ステージ226のリソースは、CPUパイプライン214のEステージが実行するサイクル数と同じだけ保持される。次に、CPUパイプライン214のAステージ246が、データバス122に、LDS命令(CPUロード・コミュニケーションレジスタ命令)で参照される“Rm”というレジスタの内容を載せる。この命令はキャッシュメモリのアクセスを要求する命令ではないから、Aステージ246はCPUレジスタファイル410のレジスタRmのデータをデータバスにロードするだけである。すなわち、ステージA(T')314はメモリアccessが起きないことを表している。

【0041】CPUパイプラインがデータバス122に

レジスタ“Rm”の内容を置いているとき、FPUパイプライン210のE2ステージ230は、文字“T”によって示されるように、どんな動作も伴わずに命令を通過させる。CPUパイプラインのAステージ246は、データレディー時間318の限られた期間、レジスタRmの内容をデータバス122上で利用出来るようにする。データレディー時間とは、CPUロード・コミュニケーションレジスタ命令に関連するデータ転送によってデータバスがビジーである時間とされる。データバス122上のデータが利用可能な前記期間318の間、FPUパイプライン210のライトバックステージSf234がデータバス122上のデータを獲得し、そのデータをレジスタFPULにストアする。

【0042】この例からも明かなように、CPUパイプライン214とFPUパイプライン210とのパイプラインステージの段数は同じである。したがって、CPU118にとってはメモリへの書込みと同様のシーケンスを採れば（実際にはメモリアクセスは行われぬ）、FPUパイプライン210はステージSfでそのデータをFPULレジスタに取り込むことができる。

【0043】CPUストア命令“STS FPUL, Rn”は、図4に示されるように、前記同様、二つのパイプライン210、214によって実行される。そのCPUストア命令は、レジスタ（FPUL）の内容をCPU汎用レジスタRnにコピーする。このCPUストア命令の場合、FPU114は、データバス122上でレジスタFPULの内容を利用可能にする期間322を制御する。すなわちFPUパイプラインはFPULレジスタの内容を322の期間だけデータバスに載せ、CPUパイプライン214はこのデータをステージSiでレジスタRnに取り込む。

【0044】図5はFPUパイプライン210の更に詳細な回路406を示す。図5の回路には、夫々FPUパイプラインステージ222、226、230、234に含まれるマスタ・スレーブ形式のラッチ414、418、422、426が示される。FPUパイプラインステージ222、226、230、234はそれらの出力をそれらに対応されるラッチ414、418、422、426に、1相目のクロック信号408の立ち下がりエッジ又は2相目のクロック410の立ち下がりエッジの何れかに同期してストアする。また、以下に記述されるバイパス信号428が示されている。FPUパイプライン回路406の機能は、すぐにそして以下で述べられる例によって示される。CPUロード・コミュニケーションレジスタ命令の例において、FPUデコードステージ（Df）222は、FPULレジスタにデータがロードされるように、マルチプレクサ434の選択信号経路430上で選択信号をアサートすることによってデータバス122からレジスタRmの内容を獲得する制御を行なう。

【0045】図6はCPU命令とFPU命令の二つの命令の実行手順におけるパイプライン210と214との同期の一例を示す。特に図6は、前述のCPUストア命令が後に続くことになる浮動小数点切り捨て及び整数へ経の変換命令510（図41及び図42参照）とを示す。図6において、一つの命令におけるFPUパイプライン210とCPUパイプライン214の夫々のパイプラインステージは、図示を簡単にするために一つの連続するブロックに統合して図示されている。だから、例えば、二つのデコードステージを示す代わりに、Df222とDi238を示す文字“D”を伴って示されている。図6の表示において時間は、クロック信号514（簡単のため、クロック信号の二つの相は示されていない）の一つの相によって示されるように、左から右に経過する。このパイプラインの表現形式は、適当なステージを単に命令が通過する状態を他と区別して表すために、パイプラインステージの指定において、文字“T”が括弧の中に挿入されている、という点を除いて、この技術分野では標準的である。例えば、浮動小数点ユニットのFPUパイプライン210の実行ステージ（E1）226が浮動小数点切り捨て命令510を実行しているとき、CPUパイプライン214の実行ステージ（E）242は、文字“T”によって示されるように何ら動作することなく単に命令を通過させる。

【0046】図6のFPU命令“FTRC Frm, FPUL”とCPU命令“STS FPUL, Rn”とを比べれば明かなように、FPULに関してデータコンフリクト（レジスタコンフリクト）を生ずる。データコンフリクトによるストールの発生を防止するために、即ち、CPUパイプライン214がストア命令を実行しているとき、CPUパイプライン214でのストールを防止するために、前記命令“FTRC”におけるCPUデコードステージ238はバイパス信号経路255にバイパス信号522をアサートする。バイパス信号522のアサートは、命令“FTRC”のE2ステージ230が浮動小数点切り捨て命令510を実行完了した後であって命令“STS”のE2ステージがストア命令を実行する準備ができたとき、当該ストア命令“STS”のE2ステージ230の入力に、前記命令“FTRC”にけるFPUパイプライン210のE2ステージ230の出力を利用可能にさせる。

【0047】図5のパイプライン回路はデータの上記バイパスを達成するようになっている。すなわち、E2ステージのラッチ422の出力はバイパス経路438で利用可能になる。バイパス経路438は、マルチプレクサ442の入力である。CPUデコードステージ238からのバイパス信号428はE2ステージのラッチ422の出力を選択し、それによって、それがE2ステージ230を通してリサイクルされることができる。浮動小数点切り捨て命令510の結果がE2ステージ230の出

カラッチ422を介して利用可能になる時間は、図6において526で示される。CPUのDiステージ238からのバイパス信号428は、マルチプレクサ422を通してデータの転送を可能にする適切な時間にアクティブになる。既に説明したように、上記ストア命令“STS”においてFPU114は、CPUレジスタRnに転送するためにデータバス122上のデータが利用可能であるところの期間322を制御する。

【0048】主に、FPUパイプライン210とCPUパイプライン214の同期化は、双方のパイプラインでストールとフリーズの効果をもたらすこれらパイプライン210、214の何れか一方がストールとフリーズの制御機能を持つことによって達成される。図2に示されるように、CPUパイプラインのデコードステージ(Di)238は、矢印255によって示されるストール信号データ経路を介してFPUパイプラインをストールする。パイプラインストールは、命令シーケンスの間、例えば、最初の命令がレジスタにライトするとき、そのレジスタの内容が次の命令によって利用される間に、起こる。前記次の命令に、更新されていないレジスタの内容を利用させないようにするために、前記最初の命令によってレジスタが更新されるまで、当該次の命令がストールされる。ここで説明する実施の態様では、そのようなストールは、一つの命令の実行ステージ(E)242の出力を次の命令の実行ステージ(E)242の入力にバイパスすることによって発生されなくなるようになっている。類似のバイパスはFPUパイプラインに存在する。ストールの間、ストールされるべきパイプラインステージ(例えばEステージ242又はE1ステージ226)にはNOP(ノン・オペレーション)が生成される。

【0049】このように、レジスタコンフリクトを生ずるとき、ストールを発生させないようにするにはバイパスを利用することができる。このとき、バイパスの制御は、図2からも明らかなように、CPUデコードステージDiが行う。すなわち、FPUパイプラインとCPUパイプラインとのパイプラインステージの段数が基本的に同一にされるから、ストール、フリーズ、バイパスを制御するフロー制御の論理はCPUパイプラインとFPUパイプラインとの間で共有することができる。この意味において、FPUパイプラインとCPUパイプラインのフロー制御のためのリソースは、双方のパイプラインで共有することができる。パイプライン制御のためのリソースを双方のパイプライン210、214で共有出来るが故に、フロー制御のハードウェアをFPUとCPUが別々に持たずに済み、CPUパイプラインのフロー制御ロジックをFPUパイプラインのフロー制御にも流用することができ、これにより、CPUパイプラインとFPUパイプラインとを同期化することが容易になる。

【0050】次に、追加のストール条件に関連する回路を例示する。図7はストールの第1のタイプすなわちロ

ード・ユース・ストールを例示するパイプラインダイアグラムである。図7には3個の命令610、614、618のシーケンスが存在する。第1の命令610は第2の命令614と同様にCPU命令である。第3番目にフェッチされる命令は浮動小数点命令である。命令610は、レジスタR2に格納されているアドレスでメモリ146の内容をレジスタR1にロードするロード命令である。第2の命令614はレジスタR1の内容をレジスタR2の内容に加算する命令である。第1の命令610はCPUパイプライン114のAステージ146でメモリ146をアクセスしており、第2の命令614はストールされる。さもなければ、第2の命令614は、命令610がレジスタR1の内容を更新している間に、レジスタR1の更新されていない内容をアクセスするであろう。ストールされると、命令614はデコードステージDから処理をやり直す。

【0051】一つのパイプラインにおけるストールは、両方のパイプライン(CPUパイプライン及びFPUパイプライン)でストールを引き起こすので、浮動小数点命令である第3の命令618は、第2の命令614とともにストールされることになる。CPUパイプライン214のDステージ238は、双方のパイプライン210、214をストールするためのストール信号622を生成する。追加のストールを防止するために、Aステージ246の内容はCPUパイプライン214のEステージ642にバイパスされている。このバイパスが行われないなら、命令610のライトバックステージSが完了されるまで、ストールされなければならない。

【0052】図8はストールのもう一つのタイプ、すなわちメモリ・アクセス・コンフリクト・ストールを例示する。図8に示されるストールは、二つの命令が同時にキャッシュメモリ146をアクセスしようとすることによって起こる。図8は4個の命令のシーケンスを実行するためのパイプライン・ダイアグラムが示される。第1の命令は図7のCPUムーブ命令610としてのCPUムーブ命令751である。図7の命令に対し、命令715はメモリ146をアクセスする。

【0053】図8において、第1の命令715は、命令フェッチステージ218により単一フェッチ動作でフェッチされた二つの命令の内の一つであり、図8では簡単のために他方の命令フェッチは図示されていない。すなわち、16ビット固定長命令に対する命令フェッチは32ビットデータバスを介して2命令単位(32ビット)で行うことができるようにされているからである。図8において、命令フェッチステージ218がメモリ146から第4番目の命令をフェッチする準備が完了するであろう時、第1の命令715はAステージ246でメモリ146をアクセスするから、Diステージ238は双方のパイプ210、214をストールする。したがって、第4番目の命令720は、ストールされない場合に比べ

て1クロックサイクル遅れてフェッチされる。

【0054】更に、第3番目の命令730は、更に以下で記述される図11の説明に含まれるストール信号1010の適用によって、D i ステージがやり直しのために繰り返される。更に、NOP（ノン・オペレーション）が、ストール信号725の適用によって、第3の命令730のE ステージ642に挿入される。そして、一方をストールすることは同様に他方をストールすることになるから、双方のパイプライン210、214がストールされる。

【0055】図9はストール信号622、725を生成する（CPUパイプライン214のD i ステージ238における）回路810のダイアグラムである。図7のストール信号622を生成するために、回路810は、ディスティネーションレジスタ（図7の命令610のR1）がソースレジスタ（図7の命令614のR1）と同一かどうかを決定するためにコンパレータ815を利用する。もしもそのような同一性があるなら、OR回路820がストール信号622を生成する。

【0056】同様に、メモリアクセス回路825がメモリアクセスのコンフリクトを決定するなら、図8について上記したように、そのとき、OR回路820はストール信号725を生成する。

【0057】図10はCPUデコードステージ238が、CPU及びFPUパイプライン210、214の双方を同時にストールすることによりどうやって同期化を維持するかについて例示する。図10において、2個の命令は連続的に実行される。最初、図3で説明したCPUロード・コミュニケーションレジスタ命令がFPU及びCPUパイプライン210、214によってフェッチされて実行される。次に、命令フェッチステージ218が浮動小数点命令910をフェッチする。（命令フェッチステージ（1）218は32ビットの命令を2個一度にフェッチする。このフェッチは偶数ワード境界で生ずる。だから、命令毎にフェッチサイクルを始める必要はない。）浮動小数点命令910はレジスタFPU Lの内容を整数値として解釈する。その浮動小数点命令910は更にその整数値を浮動小数点数に変換する。最後に、浮動小数点命令910は浮動小数点数を浮動小数点レジスタFR nに格納する。

【0058】図10において、浮動小数点命令910は、双方の実行ステージ（E1）226及び（E）242にストール信号914を用いるCPU118のデコードステージ（D i）238の結果として、FPU及びCPUパイプライン210、214をストールさせる。すなわち、双方のパイプライン210、214の実行ステージ226、242がストールされる。そのようなストールは文字“X”918によって共通に表される。このストールはここでは適切であり、それは、浮動小数点命令910が普通に実行準備完了されるであろうとき、命

令“LDS Rm, FPU L”を実行するCPU118のAステージ246はデータバス122上のデータを未だ利用可能にしていないからである。1サイクルストールされることにより、浮動小数点命令910はストール信号914がローレベルにされる途端に処理を継続する。命令“LDS Rm, FPU L”を実行するCPUパイプライン214のAステージ246の出力において利用可能なデータはバイパス経路922を経由してデータバス122にバイパスされ、そこから命令910を実行するFPUパイプライン210のE1ステージ226の inputs にバイパスされる。このバイパス922は更にストールを追加することを防止する。なぜならば、浮動小数点命令910は、データバス122上で利用可能なFPU Lレジスタの値を処理することによって命令を実行することができるからである。

【0059】同様に、この実施の態様において、パイプライン・フリーズが一つのパイプラインで起きるとき、それは双方のパイプライン（FPU及びCPUパイプライン210、214）のフリーズを同時に引き起こす。パイプラインフリーズの間、パイプライン210及び214における全ての動作は停止する。双方のパイプライン210、214のフリーズは、例えばキャッシュミスの結果として生ずる。キャッシュミスは、例えば、キャッシュメモリ146に存在しないデータをCPU118が要求したとき生ずる。その場合、キャッシュコントローラ142はCPU118にキャッシュミスが生じたことを示す信号を送る。また、CPU命令が結果を要求する乗算処理をIMAC120が完了していないとき、そのIMAC120はFPU114と同様にCPU118にもビジー信号を送り、それによってパイプラインフリーズを生ずる。

【0060】図11はFPUパイプライン210のデコードステージD f 222の回路ダイアグラムである。デコードステージ（D f）222の機能はCPU118のデコードステージ（D i）238によって生成されるストール信号1110（図11参照）に依存する。このストール信号1110は、FPUパイプライン210の実行ステージ（E1）226で利用され、E1ステージ226でNOPを生成する。D f ステージ222がストールの条件を検出すると、図11のD f ステージ222はD f の出力1014からのD f 222でデコードされた命令を帰還する。経路1018を介する帰還は、マルチプレクサ1008の入力1006（セクタ=1）を選択するためのD i ステージ238からの制御信号1010によって達成される。

【0061】図11はまた、フリーズ信号選択経路1022を示す。上述のように、フリーズ信号は全部のパイプラインで全ての処理を停止させる。同様に、フリーズ信号は、上述したように、双方のパイプライン210、214に適用される。フリーズ信号1022はラッチ1

024をディスエーブルにする。また、図11にはキャンセル信号選択経路1026が示されている。キャンセル信号選択経路1026でキャンセル信号を選択することは、マルチプレクサ(MUX)がNOP1030を挿入することであり、これにより、パイプラインのそのポイント(ステージ)でどの命令もキャンセルできる。1034で示される部分は更にFPUパイプライン210のデコードステージ222に適用される信号を示す。

【0062】1034で示される部分の記述において、シンボルで示される信号(例えば信号A, 信号B)の説明はC言語標記である。1034で示される部分において、信号名の定義における垂直線は論理的な“OR”(論理和)を示す。“&”は論理的な“AND”(論理積)を示す。“~”は論理的な反転を示す。

【0063】図12はFPUパイプライン210の第1の実行ステージ(E1)226のダイヤグラムを示す。ストールが第1の実行ステージ226で適切であるとき、上述のように、CPU118のデコードステージ238はストール信号パス1110にストール信号を適用する。これはFPUデコードステージ222の出力1038がE1ステージに渡されるのを妨げる。代わりに、経路1110でストール信号をアサートすることにより、E1ステージ226に挿入されるべきNOP1114が生成される。図12におけるフリーズ及びキャンセル信号の応用は、図11のそれと同様である。それらの信号については1114で示される部分に更に詳述される。

【0064】図13はFPUパイプライン210のステージ(Df)222, (E1)226, (E2)230, 及び(Sf)234に関する詳細な回路ダイヤグラムである。FPUパイプライン回路1208は幾つかの入力を示す。入力1210は第1オペランドのためであり、入力1214はFPU又はCPU命令の第2オペランドのためである。入力1218は、E2ステージ230の出力をE1ステージの入力に戻すためのバイパスデータを受け入れる。入力1222はデータバス(SD\_2)122の内容をE1ステージ226の入力にバイパスするためである。入力1226は浮動小数点レジスタファイルのFR0レジスタから入力である。夫々部分的にハッチングが付されている複数のラッチは、クロック信号の2相目に同期して動作される。上述の説明と図13に記述された名称に基づけば、当業者は回路1208の残りの部分を容易に理解することができるであろう。

【0065】FPUとCPUパイプライン210, 214の間における同期化に利用される命令シーケンスの他の種類としては、CPU分岐命令が後続される浮動小数点比較命令がある。図14はそのようなシーケンスを示す。浮動小数点比較命令は、浮動小数点レジスタFRm1318の値が浮動小数点レジスタFRn1322の値に等しいとき、1の値をTビットにセットする。Tビッ

トの値は、分岐先へのジャンプが実行されるか否かを決定する。分岐命令1314はCPU命令である。Tビットが1なら、分岐命令は分岐先から新たな命令のフェッチを生じさせる(この例の場合にはキャッシュメモリ146からのフェッチとなる)。図14において、Tビットの値はFPUパイプライン210のE1ステージ226からCPUパイプライン214のEステージ242にバイパスされる。換言すれば、浮動小数点比較命令を実行しているとき、Tビットの出力はE1ステージで行われる。Tビットのそのような先方へのバイパスは、浮動小数点比較命令のの後に続くCPU分岐命令1314の実行におけるストールを防止する。TビットがE2ステージで出力されるなら、1サイクルのストールが必要になってしまうからである。

【0066】図15はE1ステージ226からEステージ242へTビットをバイパスするための回路を示す。さらに、図15の回路はCPUパイプラインのステージの間でTビットをバイパスする能力を持つ。図14のバイパスの一例に対応させて説明すると、最初の命令例えば浮動小数点比較命令1310がTビット1410の値を1にセットしたとき、FPUは、E1ステージ226からTビットをマルチプレクサ1418を介して選択するために、Tビット選択パス1414を活性化する。選択回路1422は、CPUパイプラインのAステージラッチ1426若しくはSiステージラッチ1430のような幾つかの可能なTビット・ソースから、又はステータスレジスタ1434からでえさえ、選択できるようにになっている。選択信号1432は、デコードステージ(Di)238から供給され、パイプラインステージで現在実行中の命令に基づく。分岐アドレス発生回路1438は、CPUが次の命令をフェッチすることができるアドレスを生成する。もちろん、E1ステージ226からのTビット1410が0の値を持っていたなら、分岐アドレス発生回路1438は、プログラムの順番において分岐命令の後にすぐに続く命令のアドレスを選択することになる。次の命令1442のアドレスは、目的とする命令をフェッチするためにCPUパイプライン214のEステージ242に進む。

【0067】命令は時々例外を引き起こす。例えば、命令が0で除算しようとしたり、不当命令コードを利用したりするときである。例外が生ずると、一般に例外ハンドラが例外を処理するための命令系列を実行する。その後、例外ハンドラは、CPU118に、例外を引き起こした命令を伴うプログラムの実行を続けるのを許容する。すなわち、例外を生じた命令を再実行する。この実施の態様において、例外は、プログラムが規定する命令実行順序に対して実際の命令実行順序を乱さないという意味において、正確である。例えば、浮動小数点命令の例外はFPUパイプライン210のE1ステージ226で検出される。CPU118がコンピュータプログラム

のオリジナルプログラムの順番を維持するとき、例外は、例外であるにもかかわらず、正確である。オリジナルプログラムの順番は、コンパイルされたコンピュータプログラムとしての、オリジナルな命令シーケンスの順番である。換言すれば、例外が正確であるとき、コンピュータプログラムの命令は、それらが純粋に連続的なシステム、すなわちパイプライン能力を持っていないものによって実行されているかのように、実行される。例外ハンドラは、例外が起こる前のプロセッサ110の状態のような、装置の状態を退避する。

【0068】双方のパイプライン210、214で共通のストールとフリーズを持つのと同様に、例外はCPU及びFPUパイプライン210、214において双方同じパイプステージ数を持つこと（パイプライン210と214の同期化）によって正確にされる。双方のパイプライン210、214におけるストールとフリーズの同期化を達成するために、長い浮動小数点命令のための追加回路が含まれている。図16は、浮動小数点命令が比較的長い時間を完了に費やすときでも、ビジー信号をアサートすることによって、浮動小数点パイプライン210のデコードステージ222が、例外を生じさせる場合にどうやって、（プログラムの記述によって規定される命令実行順序に対し）実際の命令の実行順序を正確に保つかを示す。ビジー信号124がアサート（例えばハイレベルにセット）されるとき、その他の浮動小数点命令は、E1ステージ226が自由にされるまでFPUパイプライン210のDfステージ222を超えて進むことはできない。ビジー信号がアサートされる間、CPU命令がフェッチされたときはCPUパイプラインを用いてその命令を実行する。一方、ビジー信号がアサートされている間、他の浮動小数点命令がフェッチされるなら、当該命令はDfステージで繰り返される。これに応ずるCPUパイプライン214もストールされる。

【0069】図16において、夫々の命令は1～6で示されるシーケンスにおいてIステージ218によってフェッチされ、FPUパイプライン210又はCPUパイプライン214によって実行される。図16は6個の命令のシーケンスを例示する。第1の命令は浮動小数点除算命令1010である。FPUパイプライン210のデコードステージDf222が一旦浮動小数点命令1510を解釈すると、Dfステージ222はビジー信号1508をアサートしてCPUのデコードに供給する。ビジー信号経路124にそのビジー信号1508がアサートされた後に続いて、2）、3）で示されるような、次のCPU命令が実行を続けることができる。しかしながら、4）で示される次のFPU命令、すなわち浮動小数点加算命令1514は、CPUパイプライン214を上述のように、ストール信号1110の適用によってストールさせようとする。FPUパイプラインステージ（E1）226、（E2）230、（Sf）234は、1）

の命令の実行を続けるが、4）の命令に代表されるように、1）の命令の後の新たな浮動小数点命令は、Dfステージから先に進むことは許されない。一旦第1の浮動小数点命令がE1ステージによって実行されるのが終了されると、FPUパイプライン210のDfステージ222はビジー信号1508をネゲートする。これによって、次の浮動小数点命令、例えば浮動小数点加算命令1514をE1ステージ226に伝達するのが許容される。

10 【0070】矢印1518は、浮動小数点除算命令1510のE1ステージの実行に要する12サイクルの後に続くFPUパイプライン210のE1ステージに浮動小数点加算命令が進むことを示す。同様に矢印1522は、浮動小数点除算命令1510のE1ステージ226における第12番目のサイクルの後に続くCPUパイプライン214のDIデコードステージ238に第5番目の命令すなわちCPU加算命令が進むことを示す。このように、通常よりも実行サイクル数の長い“FDIV”のような浮動小数点命令が実行される場合にも、前記ビ  
20 ジー信号によって適切にストールされることにより、CPUパイプラインとFPUパイプラインは同期を保つことができる。仮に、浮動小数点加算命令1514の実行によって例外が発生される場合には、浮動小数点加算命令1514は13サイクル目1526で例外を発生させることになる。したがって、その例外はプログラムの実行順序を乱さない。即ち、その例外は正確である。なぜならば、浮動小数点加算命令1514の次の命令は実行の機会をまだ持っていないからである。すなわち、13  
30 サイクル目で浮動小数点加算命令1514が例外を生じたとき、後続の命令は、まだ実行ステージに進んでいないからである。これも、CPUパイプラインとFPUパイプラインが同期化を保っているからである。

【0071】図17はビジー信号経路124にビジー信号1508をアサートするためのビジー信号回路を示す。特に、最初の命令がFPUパイプライン210のデコードステージ（Df）222に入力されたという事実を示す信号は論理的なアンドゲート（AND）1618に適用される。更には、デコードステージ（Df）222は、最初の浮動小数点命令、この例の場合浮動小数点除算命令がE2ステージに入力されたという事実を示す信号を適用する。しかしながら、その信号は、アンドゲート1618によって処理される前に、インバータ1626によって反転される。そして、最初の浮動小数点命令がE1ステージ226に入力されたという事実を示す信号は、論理的なオアゲート（OR）1638の入力1634に適用される。同様に、アンドゲート1618の出力も同様にオアゲート1650の他方の入力1646に適用される。オアゲート1638の出力1650はCPUパイプライン214のデコードステージ（Di）238にビジー信号1508を供給し、それによって、C



P Uパイプライン 2 1 4 にストールを発生させることができると共に、F P Uパイプライン 2 1 0 の D f ステージで命令を繰り返すことができる。

【0072】以上本発明者によってなされた発明を実施形態に基づいて具体的に説明したが、本発明はそれに限定されるものではなく、その要旨を逸脱しない範囲において種々変更可能であることは言うまでもない。例えば、C P U と F P U のパイプライン段数は夫々 5 段に限定されず適宜変更可能である。また、本発明は R I S C プロセッサに限定されず C I S C プロセッサなど、その他のアーキテクチャを有するマイクロプロセッサに適用可能である。また、この明細書においてマイクロプロセッサは、マイクロコンピュータ、シングルチップマイクロコンピュータ、データプロセッサを含む概念として用いている。

#### 【0073】

【発明の効果】本願において開示される発明のうち代表的なものによって得られる効果を簡単に説明すれば下記の通りである。

【0074】すなわち、本発明に係るマイクロプロセッサは、命令実行に際して、F P UパイプラインとC P Uパイプラインを並列的に進行させ、浮動小数点パイプラインとC P Uパイプラインとのパイプライン段数同一とすることによって、C P UパイプラインとF P Uパイプラインのフロー制御などのパイプライン制御のためのリソースを共有させることができる。このリソースの共有化は、パイプライン制御のための回路規模を縮小可能にする。これらは、浮動小数点パイプラインとプロセッサパイプラインとを同期化し易くする。

【0075】パイプラインに対してフリーズやストールを必要とする場合、双方のパイプラインにストールとフリーズの効果をもたらすようにパイプラインの一方がストール又はフリーズを制御することによってF P UパイプラインとC P Uパイプラインの同期化を達成することができる。

【0076】F P U LレジスタのようなレジスタをC P UとF P Uが共有することによってC P UとF P Uとの間のデータ交換を簡単に且つ高速に行なうことができる。

【0077】浮動小数点比較命令の後に続く条件分岐に対するスルーブットを向上させることができる。

【0078】浮動小数点除算命令においてもF P UパイプラインとC P Uパイプラインとを同期化できる。

#### 【図面の簡単な説明】

【図 1】本発明によるマイクロプロセッサの一例を示すブロック図である。

【図 2】F P U と C P U のパイプラインの関係を示す説明図である。

【図 3】命令 “LDS Rm, FPUL” の実行に際してのF P Uパイプライン及びC P Uのパイプライン、そ

して双方のパイプライン間でのデータ交換のタイミングを示す説明図である。

【図 4】命令 “STS FPUL, Rn” の実行に際してのF P Uパイプライン及びC P Uのパイプライン、そして双方のパイプライン間でのデータ交換のタイミングを示す説明図である。

【図 5】F P U のパイプラインを更に詳細に示す説明図である。

【図 6】パイプラインリソースの共有によるF P UパイプラインとC P Uパイプラインの同期の一例を示す説明図である。

【図 7】ストールの第 1 の形態であるロード・ユース・ストールを例示するパイプラインダイアグラムである。

【図 8】ストールのもう一つの形態であるメモリアクセス・コンフリクト・ストールを例示するパイプラインダイアグラムである。

【図 9】ストール信号を生成する回路を例示するブロック図である。

【図 10】F P UパイプラインとC P Uパイプラインの双方をストールすることによりC P Uデコードステージがどうやって同期を維持するかを例示する説明図である。

【図 11】F P Uパイプラインのデコードステージを例示する説明図である。

【図 12】F P Uパイプラインの第 1 実行ステージ E 1 を例示する説明図である。

【図 13】F P U の詳細回路を例示するブロック図である。

【図 14】C P U分岐命令が後に続く浮動小数点比較命令の手順のためのF P UとC P Uパイプラインの同期化を例示する説明図である。

【図 15】E 1 ステージから E ステージに T ビットをバイパスするための回路の一例を示すブロック図である。

【図 16】例外を正確に保つためのF P UとC P Uのパイプラインの同期化の一例を示す説明図である。

【図 17】ビジー信号経路にビジー信号をアサートするためのビジー信号回路の一例を示すブロック図である。

【図 18】レジスタコンフリクトの説明図である。

【図 19】C P UパイプラインとF P Uパイプラインのリソース共有状態の一例を概念的に示す説明図である。

【図 20】図 1 のマイクロプロセッサがサポートする命令セットの内、浮動小数点命令と浮動小数点命令に関連するC P U命令とを列挙した説明図である。

【図 21】浮動小数点命令 “FABS” の説明図である。

【図 22】浮動小数点命令 “FADD” の説明図である。

【図 23】図 22 に続く浮動小数点命令 “FADD” の説明図である。

【図 24】図 23 に続く浮動小数点命令 “FADD” の



説明図である。

【図25】浮動小数点命令“FCMP”の説明図である。

【図26】図25に続く浮動小数点命令“FCMP”の説明図である。

【図27】図26に続く浮動小数点命令“FCMP”の説明図である。

【図28】浮動小数点命令“FDIV”の説明図である。

【図29】図28に続く浮動小数点命令“FDIV”の説明図である。

【図30】浮動小数点命令“FLDIO”の説明図である。

【図31】浮動小数点命令“FLDI1”の説明図である。

【図32】浮動小数点命令“FLDS”の説明図である。

【図33】浮動小数点命令“FMUL”の説明図である。

【図34】図33に続く浮動小数点命令“FMUL”の説明図である。

【図35】浮動小数点命令“FNEG”の説明図である。

【図36】浮動小数点命令“FSQRT”の説明図である。

【図37】浮動小数点命令“FSTS”の説明図である。

【図38】浮動小数点命令“FSUB”の説明図である。

【図39】図38に続く浮動小数点命令“FSUB”の説明図である。

【図40】図39に続く浮動小数点命令“FSUB”の説明図である。

【図41】浮動小数点命令“FTRC”の説明図である。

【図42】図41に続く浮動小数点命令“FTRC”の説明図である。

【図43】浮動小数点命令“FTST”の説明図である。

【図44】CPU命令“LDS”の説明図である。

【図45】図44に続くCPU命令“LDS”の説明図である。

【図46】CPU命令“STS”の説明図である。

【図47】図46に続くCPU命令“STS”の説明図である。

【図48】浮動小数点命令“FLOAT”の説明図である。

【図49】浮動小数点命令“FMAC”の説明図である。

【図50】図49に続く浮動小数点命令“FMAC”の説明図である。

【図51】浮動小数点命令“FMOV”の説明図である。

【図52】図51に続く浮動小数点命令“FMOV”の説明図である。

#### 【符号の説明】

110 マイクロプロセッサ

114 FPU

118 CPU

210 FPUパイプライン

214 CPUパイプライン

218 命令フェッチステージ (I)

222 FPUパイプラインのデコードステージ (Df)

226, 230 FPUパイプラインの実行ステージ (E1, E2)

234 FPUパイプラインのライトバックステージ (Sf)

238 CPUパイプラインのデコードステージ (Di)

242 CPUパイプラインの実行ステージ (E)

246 CPUパイプラインのメモリアクセスステージ (A)

250 CPUパイプラインのライトバックステージ (Si)

255 バイパス、ストール、キャンセル経路

FPU L コミュニケーションレジスタ

428 CPUデコードステージからのバイパス信号

518 CPUデコーダからのバイパス信号

622 ストール信号

725 ストール信号

914 ストール信号

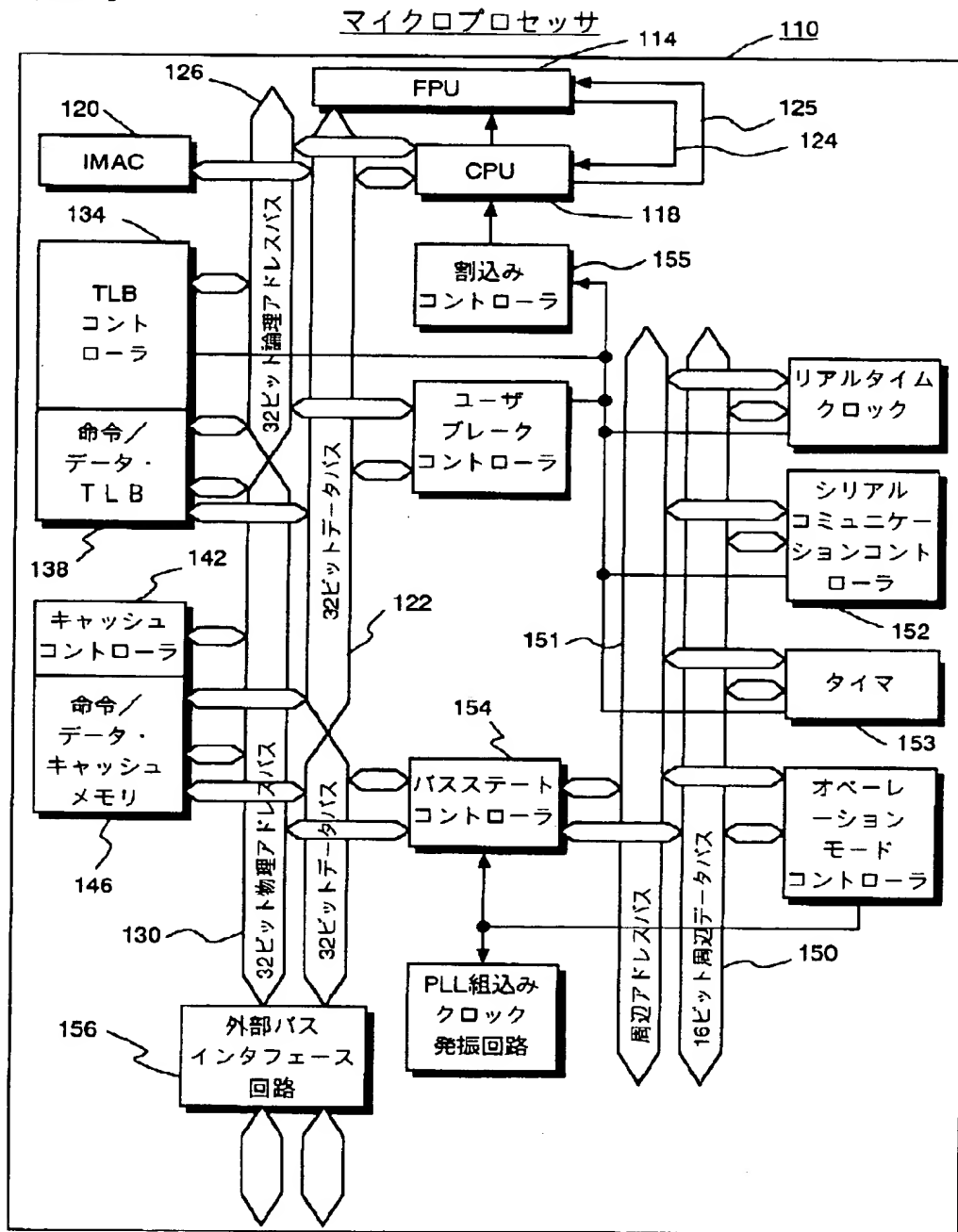
1410 E1ステージからのTビット

1508 ビジー信号

1608 ビジー信号

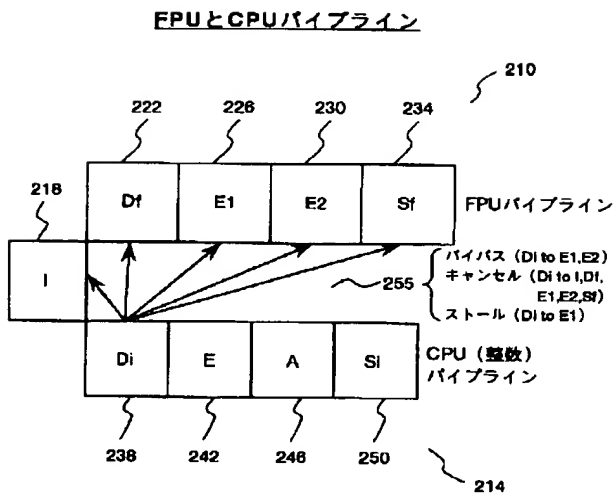
【図1】

【図1】



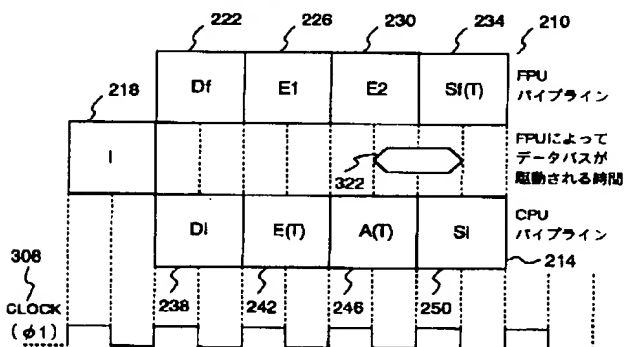
【図2】

【図2】

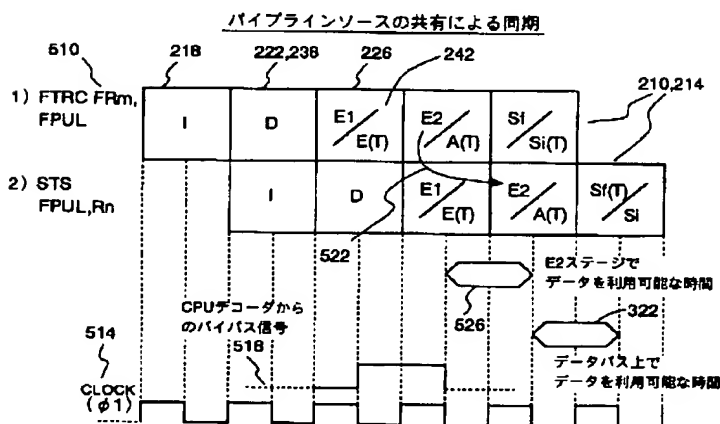


【図4】

【図4】

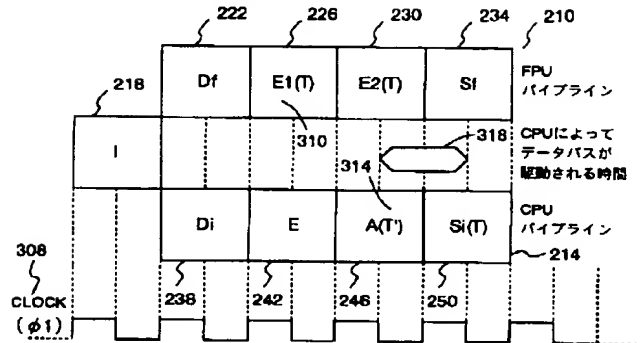
**STS FPUL, Rn**

【図6】



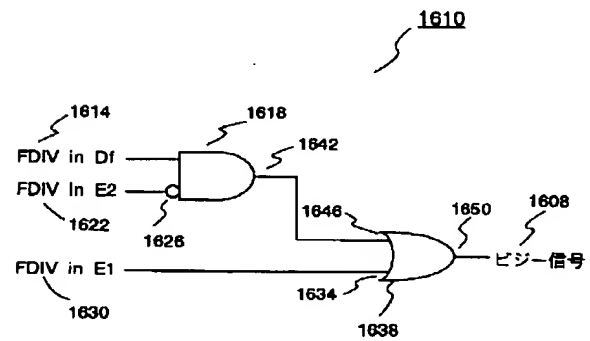
【図3】

【図3】

**LDS Rm, FPUL**

【図17】

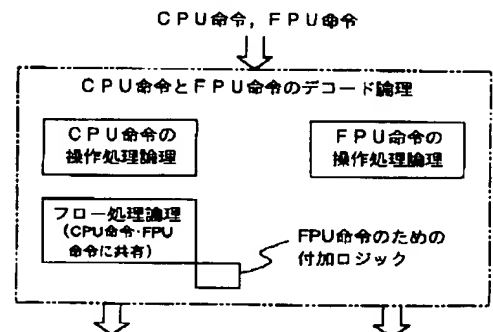
【図17】

**ビジー信号回路**

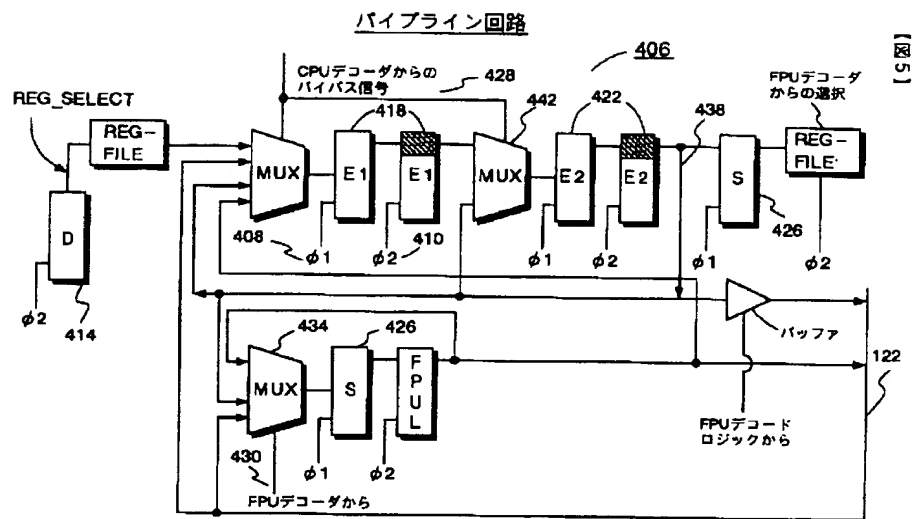
【図19】

【図19】

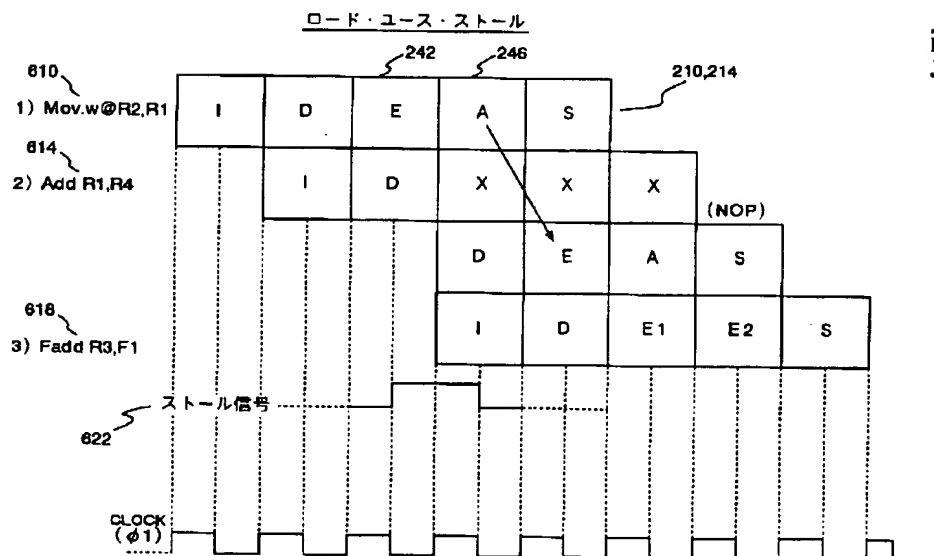
【図19】



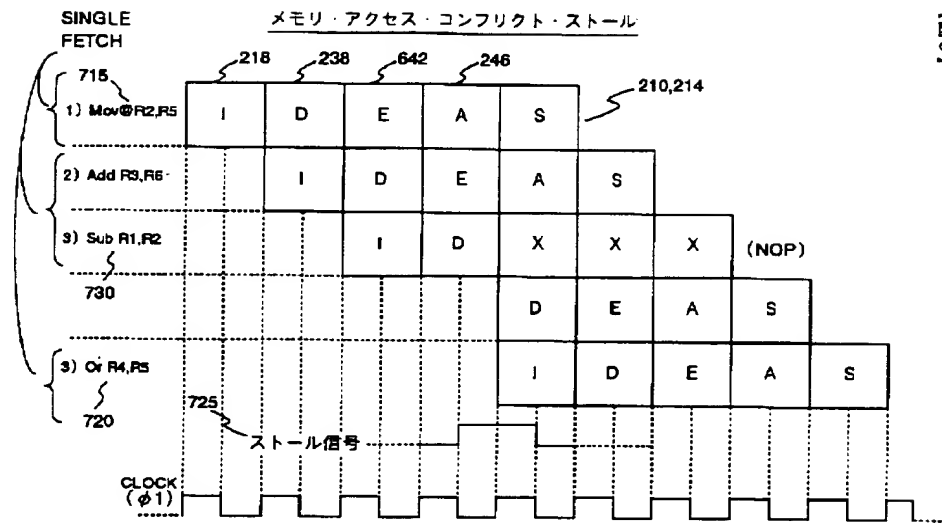
【図5】



【図7】

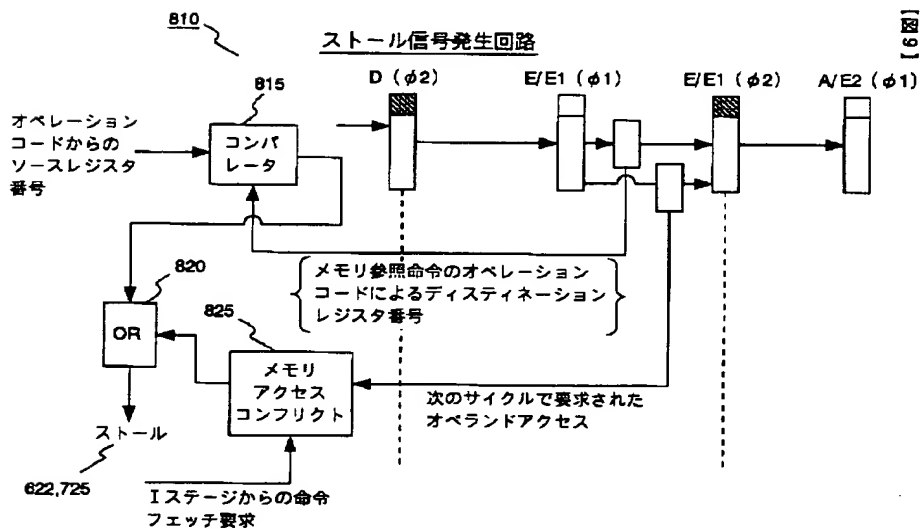


【図8】



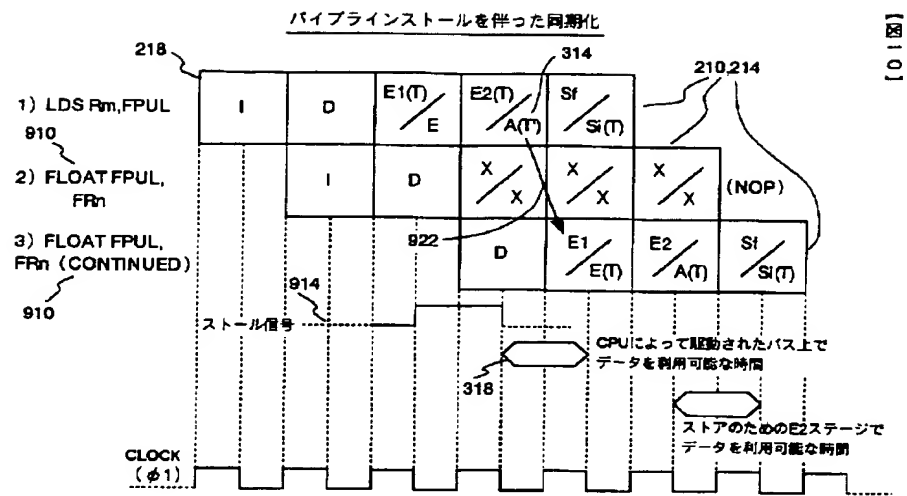
【図8】

【図9】

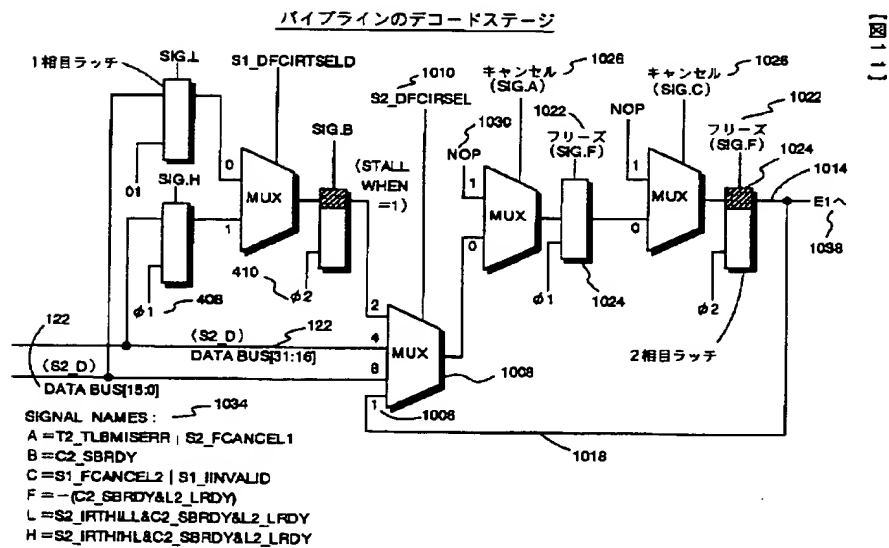


【図9】

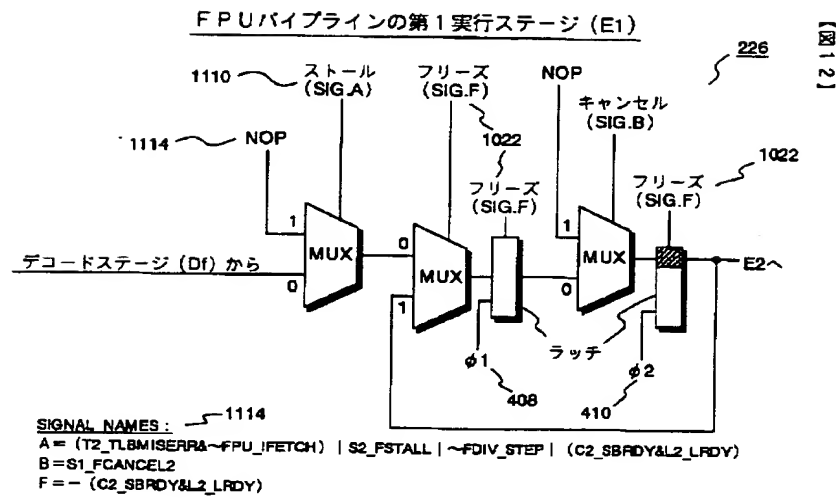
【※ 10】



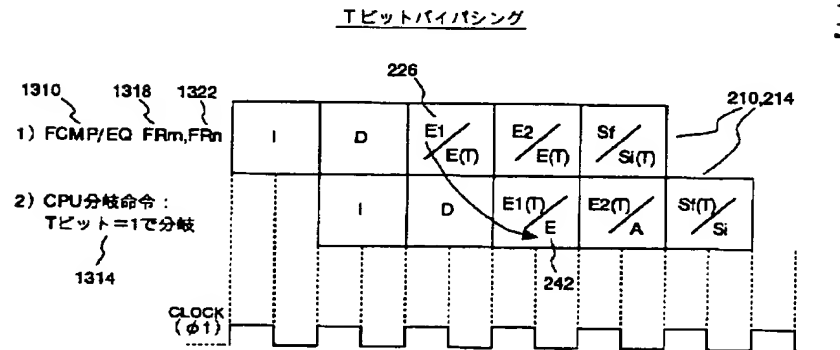
【图 1 1】



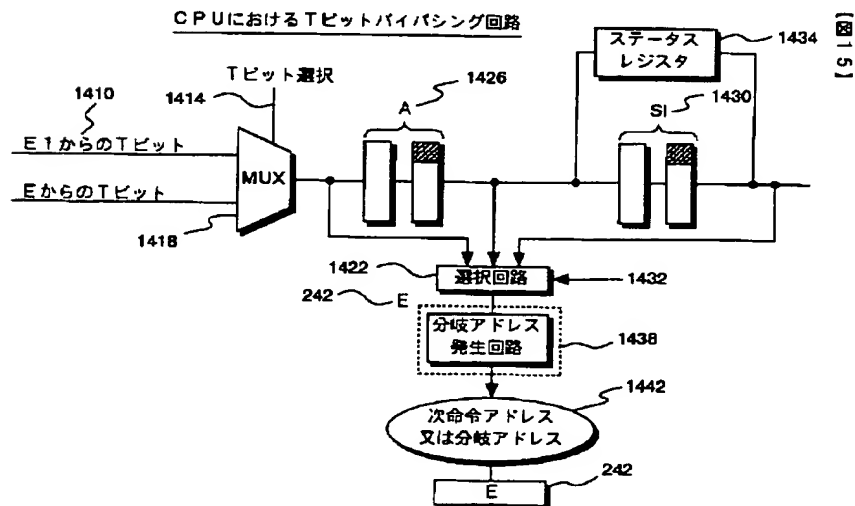
【図12】



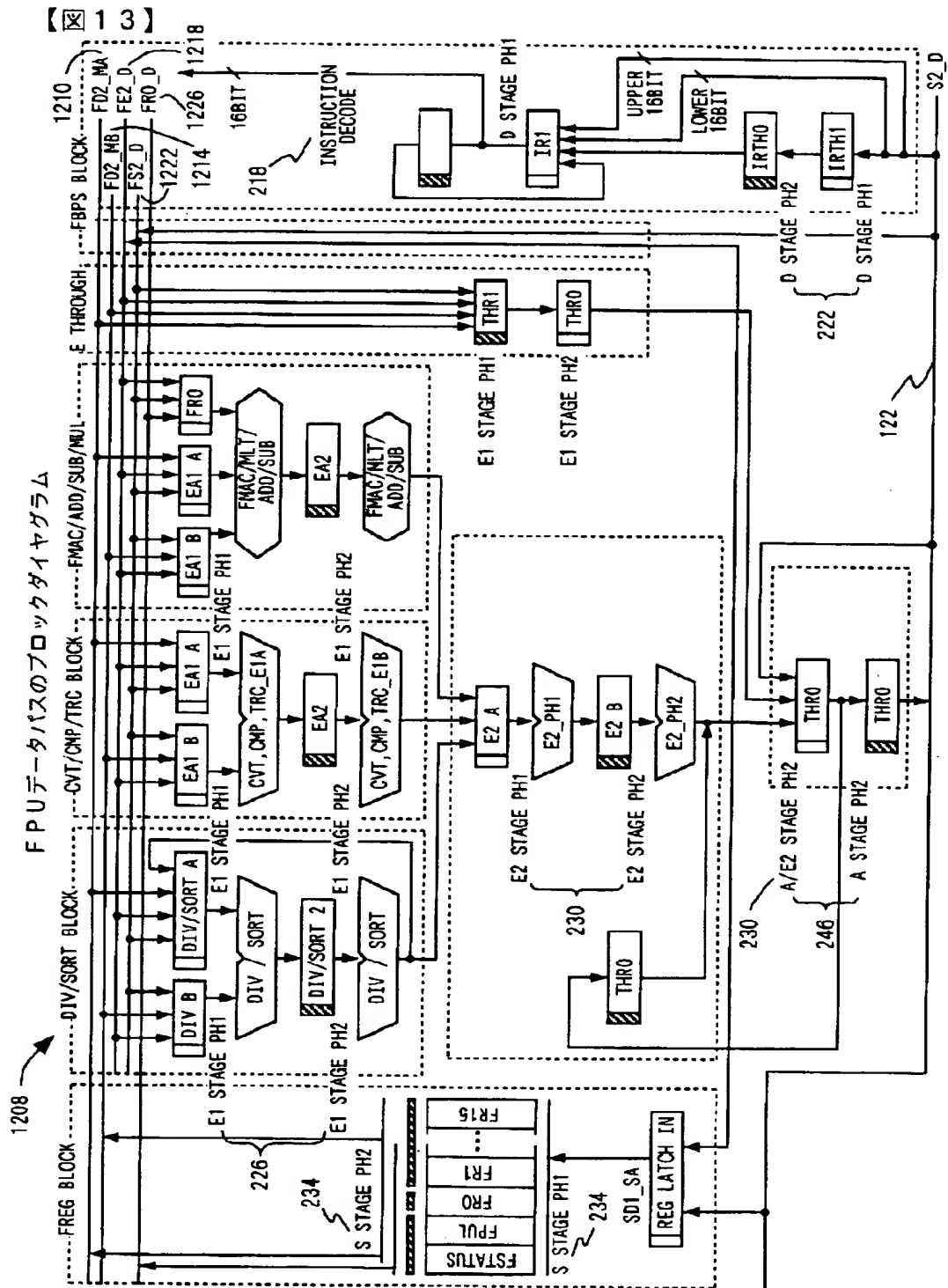
【図14】



【図15】

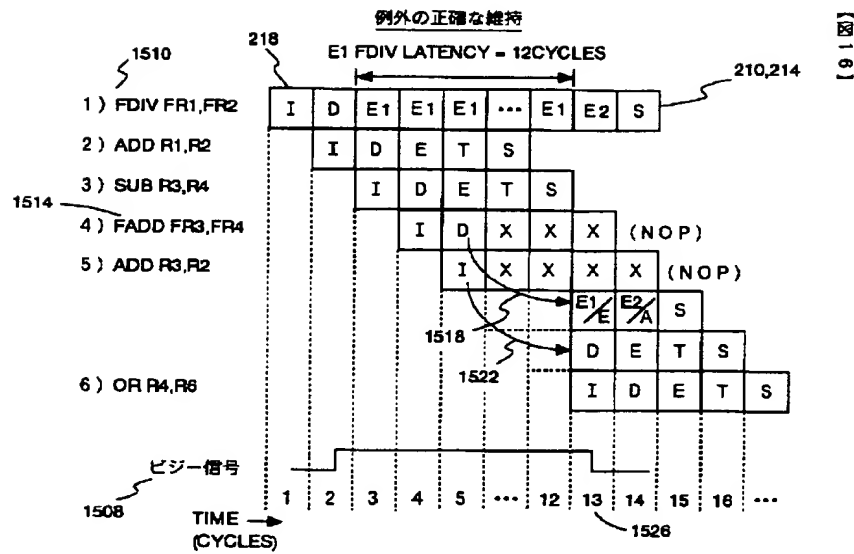


【図 13】

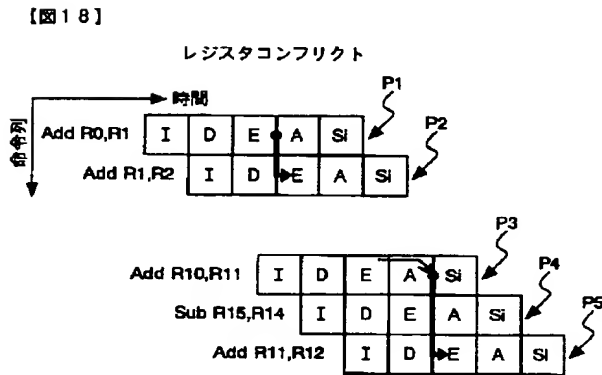




【図 16】



【図 18】



【図 20】

【図 20】

浮動小数点命令

Operation	op code	mnemonic
Floating Move (Load)	FNMF	FMOV.S @Rm,EBn
Floating Move (Store)	FNMA	FMOV.S EBn,@Rn
Floating Move (Fload)	FNMF	FMOV.S @Rm*,EBn
Floating Move (Fstore)	FNMA	FMOV.S EBn,@Rn*
Floating Move (Load with index)	FNMF	FMOV.S @R0,Rm,EBn
Floating Move (Store with index)	FNMA	FMOV.S EBn,@R0,Rn
Floating Move (n register file)	FNMC	FMOV EBm,EBn
Floating Load Immediate 0	FN80	FLDI0 EBn
Floating Load Immediate 1	FN90	FLDI1 EBn
Floating Add	FNMD	FADD EBm,EBn
Floating Subtract	FNMF	FSUB EBm,EBn
Floating Multiply	FNMD	FMUL EBm,EBn
Floating Divide	FNMF	FDIV EBm,EBn
Floating Multiply Accumulate	FNME	FMAC EB0,EBm,EBn
Floating Compare Equal	FNMF	FCMP/EQ EBm,EBn
Floating Compare Greater Than	FNMF	FCMP/GT EBm,EBn
Floating Test NaN	FN70	FTSTL NaN EBn
Floating Negate	FN40	FNEG EBn
Floating Absolute Value	FN50	FABS EBn
Floating Square Root	FN60	FSQRT EBn
Floating Convert from Integer	FN20	FLOAT FPUL,EBn
Floating Truncate and Convert to Integer	FN30	FTRC EBm,FPUL
Floating Store from System Register FPUL	FN00	FSTS FPUL,EBn
Floating Load to System Register FPUL	FN10	FLDS EBm,FPUL

FPU関連CPU命令

Operation	op code	mnemonic
Load from System Register FPUL	4N5A	LDS Rm,FPUL
Store to System Register FPUL	4N56	LDS.L @Rm*,FPUL
Load from System Register FPSCR	4N6A	LDS Rm,FPSCR
Store to System Register FPSCR	4N66	LDS.L @Rm*,FPSCR
Store to System Register FPUL	0N5A	STS FPUL,Rn
Store to System Register FPUL	4N52	STS.L FPUL,@Rn
Store to System Register FPSCR	0N6A	STS FPSCR,Rn
Store to System Register FPSCR	4N62	STS.L FPSCR,@Rn

【図21】

【図21】

FABS (Floating Point Absolute Value): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FABS $ER_n$	$ ER_n  \rightarrow ER_n$	1111nnnn01011101	2	1	-

Description: Takes floating-point-arithmetic absolute value of the content of floating point register  $ER_n$ . And the result of this operation is written on the  $ER_n$ .

Operation:

```
FABS(float *ERn) /* FABS  $ER_n$  */
{
    clear_cause_VZ();
    case(data_type_of(ERn)) {
        NORM : if(sign_of(ERn) == 0) *ERn = *ERn;
              else *ERn = -*ERn;
              break;
        PZERO:
        NZERO: zero(ERn, 0); break;
        PINF :
        NINF : inf(ERn, 0); break;
        qNaN : qnan(ERn); break;
        sNaN : invalid(ERn); break;
    }
    pc += 2;
}
```

FABS Special Cases

$ER_n$	NORM	+0	-0	+INF	-INF	qNaN	sNaN
FABS( $ER_n$ )	ABS	+0	+0	+INF	+INF	qNaN	Invalid

Denormalized value is treated as ZERO.

Exceptions:  
Invalid operation



## 【図 2 3】

## 【図 2 3】

```

NZERO:
  case(data_type_of(ERn))
  {
    NORM :      *ERn = *ERn + *ERm;      break;
    PZERO:      zero(ERn, 0);             break;
    NZERO:      zero(ERn, 1);             break;
    PINF :      inf(ERn, 0);              break;
    NINF :      inf(ERn, 1);              break;
  }
  break;
PINF :
  case(data_type_of(ERn))
  {
    NINF :      invalid(ERn);             break;
    default:    inf(ERn, 0);              break;
  }
  break;
NINF :
  case(data_type_of(ERn))
  {
    PINF :      invalid(ERn);             break;
    default:    inf(ERn, 1);              break;
  }
  break;
}
pc += 2;
}

```

## 【図 2 4】

## 【図 2 4】

FADD Special Cases							
ERm	ERn						
	NORM	+0	-0	+INF	-INF	qNaN	sNaN
NORM	ADD				-INF		
+0		+0					
-0		-0					
+INF				+INF	Invalid	qNaN	Invalid
-INF	-INF			Invalid	-INF		
qNaN							
sNaN							

Denormalized value is treated as ZERO.

Exceptions:

Invalid operation

【図25】

【図25】

## FCMP (Floating Point Compare): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FCMP/EQ <u>ERm</u> , <u>ERn</u>	( <u>ERn</u> == <u>ERm</u> ) ? 1:0	->T 1111nnnnnnnnnn0100	2	1	1/0
FCMP/GT <u>ERm</u> , <u>ERn</u>	( <u>ERn</u> > <u>ERm</u> ) ? 1:0	->T 1111nnnnnnnnnn0101	2	1	1/0

Description: Floating-point-arithmetically compares between the contents of floating point registers ERm and ERn.

And the result of this operation, true/false, is written on the T bit.

Operation:

```

FCMP_EQ(float *ERm, *ERn) /* FCMP/EQ ERm, ERn */
{
    clear_cause_VZ();
    if(fcmp_chk(ERm, ERn) == INVALID) ( fcmp_invalid(0); )
    else if(fcmp_chk(ERm, ERn) == EQ)   T = 1;
    else                                T = 0;
    pc += 2;
}

FCMP_GT(float *ERm, *ERn) /* FCMP/GT ERm, ERn */
{
    clear_cause_VZ();
    if(fcmp_chk(ERm, ERn) == INVALID) ( fcmp_invalid(0); )
    else if(fcmp_chk(ERm, ERn) == GT)   T = 1;
    else                                T = 0;
    pc += 2;
}

fcmp_chk(float *ERm, *ERn)

```

【図26】

【図26】

```

{
    if((data_type_of(FRm) == sNaN) ||
        (data_type_of(FRn) == sNaN)) return(INVALID);
    else if((data_type_of(FRm) == qNaN) ||
        (data_type_of(FRn) == qNaN)) return(UO);
    else case(data_type_of(FRm)) {
        NORM : case(data_type_of(FRn)) {
            PINF : return(GT); break;
            NINF : return(NOTGT); break;
            default: break;
        }
        PZERO:
        NZERO: case(data_type_of(FRn)) {
            PZERO: break;
            NZERO: return(EQ); break;
            PINF : return(GT); break;
            NINF : return(NOTGT); break;
            default: break;
        }
        PINF : case(data_type_of(FRn)) {
            PINF : return(EQ); break;
            default: return(NOTGT); break;
        }
        NINF : case(data_type_of(FRn)) {
            NINF : return(EQ); break;
            default: return(GT); break;
        }
    }
    if(*FRn==*FRm) return(EQ);
    else if(*FRn>*FRm) return(GT);
    else return(NOTGT);
}
fcmp_invalid(int cmp_flag)

```

【図27】

【図27】

```
set_v():  
  if((FPSCR & ENABLE_V) == 0) T = cmp_flag;  
}
```

FCMP Special Cases

ERn						
ERm	NORM	+0	-0	+INF	-INF	sNaN
NORM	CMP	EQ	EQ	GT	! GT	qNaN
+0						
-0						
+INF	! GT			EQ	EQ	UO
-INF						
qNaN	GT					
sNaN						Invalid

Denormalized value is treated as ZERO.

Exceptions:  
Invalid operation

Note: IEEE defines the independly 4 conditions of caparison. But FPU support FCMP/EQ and FCMP/GT only. But FPU can support all conditions using the combination of BT/BF, FTST/NAN and these 2 FCMPs.

- Unorder ERm, ERn      fts/nan ERm : bf : fts/nan ERn ; bf
- (ERm == ERn)      fcmp/eq ERm, ERn ; bt
- (ERm != ERn)      fcmp/eq ERm, ERn ; bf
- (ERm < ERn )      fcmp/gt ERm, ERn ; bt
- (ERm <= ERn)      fcmp/gt ERn, ERm ; bf
- (ERm > ERn)      fcmp/gt ERn, ERm ; bt
- (ERm >= ERn)      fcmp/gt ERm, ERn ; bf

【図28】

【図28】

FDIV (Floating Point Divide): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FDIV $ERn, ERn$	$ERn/ERn \rightarrow ERn$	1111nnnnnnnn0011	13	12	-

Description: Floating-point-arithmetically divides the content of floating point register  $ERn$  by the content of floating point register  $ERn$ . And the result of this operation is written on the  $ERn$ .

Operation:

```
FDIV(float *ERn, *ERn) /* FDIV  $ERn, ERn$  */
{
    clear_cause_VZ();
    if((data_type_of(ERn) == sNaN) ||
        (data_type_of(ERn) == sNaN)) invalid(ERn);
    else if((data_type_of(ERn) == qNaN) ||
        (data_type_of(ERn) == qNaN)) qnan(ERn);
    else case(data_type_of(ERn)) {
        NORM:
            case(data_type_of(ERn)) {
                PINF:
                    NINF: inf(ERn, sign_of(ERn)^sign_of(ERn)); break;
                    default: *ERn = *ERn / *ERn; break;
            }
                PZERO:
                NZERO:
                    case(data_type_of(ERn)) {
                        PZERO:
                            NZERO: invalid(ERn); break;
                            default: dz(ERn, sign_of(ERn)^sign_of(ERn)); break;
                    }
            }
    }
}
```



【図29】

```

PINF :
NINF :
    case(data_type_of(FRn))
    {
        PINF :
        NINF :  invalid(FRn);
        default:  zero(FRn, sign_of(FRn)^sign_of(FRn));
    }
    break;
    break;
}
pc += 2;

```

【図29】

FDIV Special Cases

	FRn						
FRm	NORM	+0	-0	+INF	-INF	qNaN	sNaN
NORM	DIV	0		INF		qNaN	Invalid
+0	DZ	Invalid		DZ			
-0							
+INF	0	+0	-0	Invalid			
-INF		-0	+0				
qNaN	qNaN						
sNaN							Invalid

Denormalized value is treated as ZERO.

Exceptions:

Invalid operation

【図34】

```

NINF :
    case(data_type_of(FRn))
    {
        PZERO:
        NZERO:  invalid(FRn);
        default:  inf(FRn, sign_of(FRn)^sign_of(FRn));
    }
    break;
    break;
}
pc += 2;

```

【図34】

FMUL Special Cases

FPU Special Cases								
FR <sub>m</sub>	FR <sub>n</sub>							
	NORM	+0	-0	+INF	-INF	qNaN	sNaN	
NORM	MUL	0		INF		qNaN	Invalid	
+0	0	+0	-0	Invalid				
-0		-0	+0					
+INF	INF	Invalid		+INF	-INF			
-INF				-INF	+INF			
qNaN	qNaN							
sNaN	qNaN							
							Invalid	

Denormalized value is treated as ZERO.

Exceptions:

Invalid operation

【図30】

【図30】

FLDIO (Floating Point Load Immediate 0): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FLDIO ERn	0x00000000 -> ERn	1111nnnn10001101	2	1	-

Description: Loads floating point zero (0x00000000) to the floating point register ERn.

Operation:

```
FLDIO(float *ERn) /* FLDIO ERn */  
{  
    *ERn = 0x00000000;  
    PC += 2;  
}
```

Exceptions:  
None

【図31】

【図31】

FLDI1 (Floating Point Load Immediate 1): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FLDI1 ERn	0x3F800000 -> ERn	1111nnnn10011101	2	1	-

Description: Loads floating point one (0x3F800000) to the floating point register ERn.

Operation:

```
FLDI1(float *ERn) /* FLDI1 ERn */  
(  
    *ERn = 0x3F800000;  
    PC += 2;  
)
```

Exceptions:  
None

【図32】

【図32】

FLDS (Floating Point Load to System Register): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FLDS <u>ERm</u> , FPUL	<u>ERm</u> -> FPUL	1111nnnn00011101	2	1	-

Description: Copies the content of floating point register ERm to the system register FPUL.

Operation:

```
FLDS(float *ERm, *FPUL) /* FLDS ERm, FPUL */  
{  
    *FPUL = *ERm;  
    pc += 2;  
}
```

Exceptions:  
None



【図35】

【図35】

FNEG (Floating Point Negate): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FNEG ERn	-ERn -> ERn	1111nnnn01001101	2	1	-

Description: Floating-point-arithmetically negates the content of floating point register ERn. And the result of this operation is written on the ERn.

Operation:

```
FNEG(float *ERn) /* FNEG ERn */
{
    clear_cause_VZ();
    case(data_type_of(ERn)) {
        qNaN : qnan(ERn); break;
        sNaN : invalid(ERn); break;
        default: *ERn = -(*ERn); break;
    }
    pc += 2;
}
```

FNEG Special Cases

ERn	NORM	+0	-0	+INF	-INF	qNaN	sNaN
FNEG(ERn)	NEG	-0	+0	-INF	+INF	qNaN	Invalid

Denormalized value is treated as ZERO.

Exceptions:  
Invalid operation

【図 3 6】

【図 3 6】

FSQRT (Floating Square Root): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FSQRT ERn	$\sqrt{ERn} \rightarrow ERn$	1111nnnn01101101	13	12	-

Description: Takes floating-point-arithmetic square root of the content of floating point register ERn. And the result of this operation is written on the ERn.

Operation:

```
FSQRT(float *ERn) /* FSQRT ERn */
{
    clear_cause_VZ();
    case(data_type_of(ERn)) (
        NORM : if(sign_of(ERn)==0)
                *ERn = sqrt(*ERn);
            else
                invalid(ERn); break;
        PZERO:
        NZERO:
        PINF : *ERn = *ERn ; break;
        NINF : invalid(ERn); break;
        qNaN : qnan(ERn); break;
        sNaN : invalid(ERn); break;
    )
    pc += 2;
}
```

FSQRT Special Cases

ERn	+NORM	-NORM	+0	-0	+INF	-INF	qNaN	sNaN
FSQRT(ERn)	SQRT	Invalid	+0	-0	+INF	Invalid	qNaN	Invalid

Denormalized value is treated as ZERO.

- Exceptions:
- Invalid operation

【図37】

【図37】

FSTS (Floating Point Store from System Register): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FSTS FPUL, ERn	FPUL -> ERn	1111nnnn00001101	2	1	-

Description: Copies the content of the system register FPUL to floating point register ERn.

Operation:

```
FSTS(float *ERn, *FPUL) /* FSTS FPUL, ERn */
{
    *ERn = *FPUL;
    pc += 4;
}
```

Exceptions:  
None



【図38】

【図38】

## FSUB (Floating Point Subtract): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FSUB $ERn, ERn$	$ERn - ERn \rightarrow ERn$	1111nnnnnnnn001	2	1	-

Description: Floating-point-arithmetically subtracts the content of floating point register  $ERn$  from the content of floating point register  $ERn$ . And the result of this operation is written on the  $ERn$ .

Operation:

```

FSUB(float *ERn, *ERn) /* FSUB ERn, ERn */
{
    clear_cause_VZ();
    if((data_type_of(ERn) == sNaN) ||
        (data_type_of(ERn) == sNaN)) invalid(ERn);
    else if((data_type_of(ERn) == qNaN) ||
        (data_type_of(ERn) == qNaN)) qnan(ERn);
    else case(data_type_of(ERn)) {
        NORM :
            case(data_type_of(ERn)) {
                PINF : inf(ERn, 0); break;
                NINF : inf(ERn, 1); break;
                default: *ERn = *ERn - *ERn; break;
            }
        PZERO:
            case(data_type_of(ERn)) {
                NORM : *ERn = *ERn - *ERn; break;
                PZERO: zero(ERn, 0); break;
                NZERO: zero(ERn, 1); break;
                PINF : inf(ERn, 0); break;
                NINF : inf(ERn, 1); break;
            }
    }
}

```

【図39】

【図39】

```

NZERO:
    case(data_type_of(ERn))
    {
        NORM :  *ERn = *ERn - *ERM;  break;
        PZERO:
        NZERO:   zero(ERn,0);         break;
        PINF :   inf(ERn,0);          break;
        NINF :   inf(ERn,1);          break;
    }
    break;
PINF :
    case(data_type_of(ERn))
    {
        NINF :   invalid(ERn);        break;
        default:   inf(ERn,1);        break;
    }
    break;
NINF :
    case(data_type_of(ERn))
    {
        PINF :   invalid(ERn);        break;
        default:   inf(ERn,0);        break;
    }
    break;
    }
pc += 2;
)

```

【図40】

【図40】

FSUB Special Cases

ERm	ERn								
	NORM	+0	-0	+INF	-INF	qNaN	sNaN		
NORM	SUB			+INF	-INF				
+0									
-0		+0							
+INF	-INF			Invalid					
-INF	+INF				Invalid				
qNaN	qNaN								
sNaN								Invalid	

Denormalized value is treated as ZERO.

Exceptions:

Invalid operation

【図41】

【図41】

FTRC (Floating Point Truncate and Convert to Integer): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FTRC <i>ERm</i> , <i>FPUL</i>	(long) <i>ERm</i> -> <i>FPUL</i>	1111nnnn00111101	2	1	-

Description: Interprets the content of floating point register *ERm* as a floating point number value and truncates it to an integer value. And the result of this operation is written to the *FPUL*.

Operation:

```
#define N_INT_RANGE 0xc7000000 /* -1.000000 * 2^31 */
#define P_INT_RANGE 0x46ffffff /* 1.fffffe * 2^30 */
```

```
FTRC(float *ERm, int *FPUL) /* FTRC ERm, FPUL */
{
    clear_cause_VZ();
    case(ftrc_type_of(ERm)) {
        NORM : *FPUL = (long)(*ERm); break;
        PINF : ftrc_invalid(0); break;
        NINF : ftrc_invalid(1); break;
    }
    pc += 2;
}

int ftrc_type_of(long *src)
{
    long abs;
    abs = *src & 0x7fffffff;
    if(sign_of(src) == 0) {
        if(abs > 0x7f800000) return(NINF); /* NaN */
        else if(abs > P_INT_RANGE) return(PINF); /* out of range, +INF */
        else return(NORM); /* +0, +NORM */
    }
    else {
        if(abs > N_INT_RANGE) return(NINF); /* out of range, +INF, NaN */
        else return(NORM); /* -0, -NORM */
    }
}
```

【図42】

【図42】

```
ftrc_invalid(long *dest, int sign)
{
    set_v();
    if((FPSCR & ENABLE_V) == 0) {
        if(sign == 0) *dest = 0x7fffffff;
        else *dest = 0x80000000;
    }
}
```

FTRC Special Cases

ERn	NORM	+0	-0	positive out of range	negative out of range	+INF	-INF	qNaN	sNaN
FTRC (ERn)	TRC	0	0	Invalid +MAX	Invalid -MAX	Invalid +MAX	Invalid -MAX	Invalid -MAX	Invalid -MAX

Denormalized value is treated as ZERO.

Exceptions:  
Invalid operation

【図 4 3】

【図 4 3】

FTST (Floating Point Test): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FTST/NAN ERn	(ERn==NAN)? 1:0 ->T 1111nnnn01111101	2	1	1	0

Description: Floating-point-arithmetically tests which the contents of floating point register ERn is NAN or not. And the result of this operation, true/false, is written on the T bit.

Operation:

```
FTST_NAN(float *ERn) /* FTST/NAN ERn */
{
    clear_cause_VZ();
    case(data_type_of(ERn)) {
        NORM :
        PZERO:
        NZERO:
        PINF :
        NINF :
        qNaN :
        sNaN :
        T = 0; break;
        T = 1; break;
        fcmp_invalid(1); break;
    }
    pc += 2;
}
```

FTST/NAN Special Cases

ERn	NORM	+0	-0	+INF	-INF	qNaN	sNaN
FTST/NAN(ERn)	T=0	T=0	T=0	T=0	T=0	T=1	Invalid

Denormalized value is treated as ZERO.

Exceptions:  
Invalid operation

【図44】

【図44】

LDS (Load to System Register): CPU Instruction

Format	Abstract	Code	Latency	Pitch	T bit
1 LDS Rm, FPUL	Rm->FPUL	0100nnnn0101010	1	1	-
2 LDS_LL @Rm+, FPUL	@Rm->FPUL, Rm+=4	0100nnnn01010110	2	1	-
3 LDS Rm, FPSCR	Rm->FPSCR	0100nnnn01101010	1	1	-
4 LDS_LL @Rm+, FPSCR	@Rm->FPSCR, Rm+=4	0100nnnn01100110	2	1	-

- Description:
- 1. Copies the content of general purpose register Rm to system register FPUL.
  - 2. Loads the content of the memory location addressed by general register Rm. And the result of this operation is written on system register FPUL. Upon successful completion, the value in Rm is incremented by 4.
  - 3. Copies the content of general purpose register Rm to system register FPSCR. The predetermined bits of FPSCR remain unchanged.
  - 4. Loads the content of the memory location addressed by general register Rm. And the result of this operation is written on system register FPSCR. Upon successful completion, the value in Rm is incremented by 4. The predetermined bits of FPSCR remain unchanged.

Operation:

```
#define FPSCR_MASK 0x00018c60

LDS(long *Rm, *FPUL) /* LDS Rm, FPUL */
{
    *FPUL = *Rm;
    pc += 2;
}

LDS_RESTORE(long *Rm, *FPUL) /* LDS_LL @Rm+, FPUL */
{
    if(load_long(Rm, FPUL) != Address_Error) *Rm += 4;
    pc += 2;
}
```

【図45】

【図45】

```

LDS(long *Rm, *FPSCR)          /* LDS Rm, FPSCR */
{
    *FPSCR = *Rm & FPSCR_MASK;
    pc += 2;
}
LDS_RESTORE(long *Rm, *FPSCR)   /* LDS_L @Rm+, FPSCR */
{
    long *tmp_FPSCR;
    if(load_long(Rm, tmp_FPSCR) != Address_Error){
        *FPSCR = *tmp_FPSCR & FPSCR_MASK;
        *Rm += 4;
    }
    pc += 2;
}

```

Exceptions:  
Address Error

【図47】

【図47】

```

{
    *Rn = *FPSCR;
    pc += 2;
}
STS_RESTORE(long *FPSCR, *Rn)   /* STS_L FPSCR, @-Rn */
{
    long *tmp_address = *Rn - 4;
    if(store_long(FPSCR, tmp_address) != Address_Error)
        Rn = tmp_address;
    pc += 2;
}

```

Exceptions:  
Address Error

【図46】

【図46】

## STS (Store from System Register): CPU Instruction

Format	Abstract	Code	Latency	Pitch	T bit
1 STS FPUL, Rn	FPUL → Rn	0100nnnn0101010	1	1	-
2 STS <sub>L</sub> FPUL, @-Rn Rn-=4, FPUL →@Rn		0100nnnn01010110	2	1	-
3 STS FPSCR, Rn	FPSCR → Rn	0100nnnn01101010	1	1	-
4 STS <sub>L</sub> FPSCR, @-Rn Rn-=4, FPSCR →@Rn		0100nnnn01100110	2	1	-

Description: 1. Copies the content of system register FPUL to general purpose register Rn.

2. Stores the content of system register FPUL into the memory location addressed by general register Rn decremented by 4. Upon successful completion, the decremented value becomes the value of Rn.

3. Copies the content of system register FPSCR to general purpose register Rn.

4. Stores the content of system register FPSCR into the memory location addressed by general register Rn decremented by 4. Upon successful completion, the decremented value becomes the value of Rn.

Operation:

```

STS(long *FPUL, *Rn) /* STS FPUL, Rn */
{
    *Rn = *FPUL;
    pc += 2;
}
STS_SAVE(long *FPUL, *Rn) /* STSL FPUL, @-Rn */
{
    long *tmp_address = *Rn - 4;
    if(store_long(FPUL, tmp_address) != Address_Error)
        Rn = tmp_address;
    pc += 2;
}
STS(long *FPSCR, *Rn) /* STS FPSCR, Rn */

```



【図 48】

【図 48】

Float (Floating Point Convert from Integer): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
Float FPUL, ERn	(float)FPUL -> ERn	1111nnnn00101101	2	1	-

Description: Interprets the content of FPUL as an integer value and converts it to a floating point number value. And the result of this operation is written on the floating point register ERn.

Operation:

```
Float(int *FPUL, float *ERn) /* Float ERn */
{
    clear_cause_VZ();
    *ERn = (float) *FPUL;
    pc += 2;
}
```

Exceptions:

None

【図49】

【図49】

FMAC (Floating Point Multiply Accumulate): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FMAC ER0, ERm, ERn, ERm*ERn->ERn		1111nnnnnnnnnn1110	2	1	-

Description: Floating-point-arithmetically multiplies the contents of floating point registers ER0 and ERm. And the result of this operation is accumulated to floating point register ERn.

Operation:

```
FMAC(float *ER0, *ERm, *ERn) /* FMAC ER0, ERm, ERn */
{
    long tmp_FPSCR;
    float *tmp_Fmul = *ERm;
    FMUL(ER0, tmp_Fmul);
    pc -= 2; /* correct pc */
    tmp_FPSCR = FPSCR; /* save cause field for ER0*ERm */
    FADD(tmp_Fmul, ERn);
    FPSCR |= tmp_FPSCR; /* reflect cause field for ER0*ERm */
}
```

【図50】

【図50】

FMAC Special Cases

FR <sub>n</sub>	FR <sub>0</sub>	FR <sub>m</sub>						qNaN	sNaN
		+NORM	-NORM	+0	-0	+INF	-INF		
NORM	NORM	MAC						qNaN	sNaN
	0								
	+INF								
	-INF								
+0	NORM							qNaN	sNaN
	0								
	+INF								
	-INF								
-0	NORM							qNaN	sNaN
	NORM								
	+0								
	-0								
	+INF								
	-INF								
	+NORM								
	-NORM								
+INF	NORM							qNaN	sNaN
	NORM								
	0								
	+INF								
	-INF								
	+NORM								
	-NORM								
	0								
-INF	NORM							qNaN	sNaN
	NORM								
	0								
	+INF								
	-INF								
	+NORM								
	-NORM								
	0								
qNaN	NORM							qNaN	sNaN
	NORM								
	0								
	+INF								
	-INF								
	+NORM								
	-NORM								
	0								
! NaN	NORM							qNaN	sNaN
	NORM								
	0								
	+INF								
	-INF								
	+NORM								
	-NORM								
	0								
all types	NORM							qNaN	sNaN
	NORM								
	0								
	+INF								
	-INF								
	+NORM								
	-NORM								
	0								
sNaN	NORM							qNaN	sNaN
	NORM								
	0								
	+INF								
	-INF								
	+NORM								
	-NORM								
	0								

Exceptions:  
Invalid operation

Denormalized value is treated as ZERO.

【図 5 1】

【図 5 1】

## FMOV (Floating Point Move): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
1 FMOV <u>FRm</u> , <u>FRn</u>	<u>FRm</u> -> <u>FRn</u>	1111nnnnnnnnnn1100	2	1	-
2 FMOV <u>S</u> , @ <u>Rm</u> , <u>FRn</u>	( <u>Rm</u> ) -> <u>FRn</u>	1111nnnnnnnnnn1000	2	1	-
3 FMOV <u>S</u> , <u>FRm</u> , @ <u>Rn</u>	<u>FRm</u> -> ( <u>Rn</u> )	1111nnnnnnnnnn1010	2	1	-
4 FMOV <u>S</u> , @ <u>Rm</u> +, <u>FRn</u>	( <u>Rm</u> ) -> <u>FRn</u> , <u>Rm</u> += 1	1111nnnnnnnnnn1001	2	1	-
5 FMOV <u>S</u> , <u>FRm</u> , @- <u>Rn</u>	<u>Rn</u> -= 4, <u>FRm</u> -> ( <u>Rn</u> )	1111nnnnnnnnnn1011	2	1	-
6 FMOV <u>S</u> , @( <u>R0</u> , <u>Rm</u> ), <u>FRn</u>	( <u>R0</u> + <u>Rm</u> ) -> <u>FRn</u>	1111nnnnnnnnnn0110	2	1	-
7 FMOV <u>S</u> , <u>FRm</u> , @( <u>R0</u> , <u>Rm</u> )	<u>FRm</u> -> ( <u>R0</u> + <u>Rn</u> )	1111nnnnnnnnnn0111	2	1	-

## Description:

1. Moves the content of floating point register FRm to the floating point register FRn.
2. Loads the content of the memory location addressed by general register Rm. And the result of this operation is written on the floating point register FRn.
3. Stores the content of floating point register FRm into the memory location addressed by general register Rn.
4. Loads the content of the memory location addressed by general register Rm. And the result of this operation is written on the floating point register FRn. Upon successful completion, the value in Rm is incremented by 4.
5. Stores the content of floating point register FRm into the memory location addressed by general register Rn decremented by 4. Upon successful completion, the decremented value becomes the value of Rn.
6. Loads the content of the memory location addressed by general register Rm and R0. And the result of this operation is written on the floating point register FRn.
7. Stores the content of floating point register FRm into the memory location addressed by general register Rn and R0.

## Operation:

```

FMOV(Float *FRm, *FRn) /* FMOV_S FRm, FRn */
(
    *FRn = *FRm;
    pc += 2;
)

```

## 【図52】

【図52】

```

FMOV_LOAD(long *Rm, float *FRn)          /* FMOV @Rm, FRn */
{
    load_long(Rm, FRn);
    pc += 2;
}
FMOV_STORE(float *FRm, long *Rn)          /* FMOV_S FRm, @Rn */
{
    store_long(FRm, Rn);
    pc += 2;
}
FMOV_RESTORE(long *Rm, float *FRn)        /* FMOV_S @Rm+, FRn */
{
    if(load_long(Rm, FRn) != Address_Error)    *Rm += 4;
    pc += 2;
}
FMOV_SAVE(float *FRm, long *Rn)            /* FMOV_S FRm, 3-Rn */
{
    long *tmp_address = *Rn - 4;
    if(store_long(FRm, tmp_address) != Address_Error)    Rn = tmp_address;
    pc += 2;
}
FMOV_LOAD_index(long *Rm, long *R0, float *FRn) /* FMOV_S @{R0, Rm}, FRn */
{
    load_long(&(*Rm+*R0), FRn);
    pc += 2;
}
FMOV_STORE_index(float *FRm, long *R0, long *Rn) /* FMOV_S FRm, @{R0, Rn} */
{
    store_long(FRm, &(*Rn+*R0));
    pc += 2;
}

```

Exceptions:

Address Error

フロントページの続き

(72)発明者 ケビン イアドナト  
 アメリカ合衆国 カリフォルニア州  
 95133、サンノゼ、ガレオン コート 678

(72)発明者 中川 典夫  
 東京都小平市上水本町五丁目20番1号 株  
 式会社日立製作所半導体事業部内  
 (72)発明者 内山 邦男  
 東京都国分寺市東恋ヶ窪一丁目280番地  
 株式会社日立製作所中央研究所内